1

Manipuler des données spatiales

Ce chapitre introduit au monde des systèmes d'informations géographiques (SIG). Il permet au novice de comprendre les concepts clés de la cartographie informatique, détaille le logiciel PostgreSQL et son fonctionnement basé sur le langage SQL et introduit à l'extension PostGIS.

1.1. Qu'est-ce qu'un Système d'Information Géographique (SIG) ?

De la cartographie aux SIG modernes

Un Système d'Information Géographique (SIG) centralise un ensemble de ressources permettant de collecter, stocker, traiter et distribuer des données intégrant une composante spatiale, c'est-à-dire géolocalisées (localisées à la surface de la Terre).

Sa fonction initiale est pratiquée depuis l'Antiquité : il s'agit de la cartographie, c'est-à-dire de la représentation du monde physique sur un support.

Ce n'est que bien plus tard, durant la première moitié du XVIII^{e} siècle, qu'apparaît l'une des principales applications des SIG modernes : les analyses spatiales. Ces dernières portaient alors à l'époque sur les effets du choléra en région parisienne.

Au début du XX^{e} siècle, plusieurs techniques sont utilisées pour séparer les différents constituants d'une carte sur plusieurs supports transparents permettant alors de les superposer et de travailler de manière plus efficace. Cette caractéristique est maintenant à la base des SIG actuels.

C'est enfin à partir du milieu du XX^{e} siècle que les SIG ont été portés sur ordinateur avec le développement de l'informatique.

Un SIG se compose des éléments suivants :

- les données : constituant principal, il s'agit d'informations géolocalisées acquises sur le terrain par des relevés GPS ou déduites depuis des photographies aériennes ;

- les infrastructures informatiques dans lesquels on retrouve :
  - la partie matérielle : serveur, ordinateur, outils de terrain (comme les GPS) ;
  - la partie logicielle : qui permet de collecter, stocker, traiter puis de redistribuer les données ;
- les méthodologies et savoir-faire : ils sont propres à chaque organisation ;
- les utilisateurs : de l'ingénieur réalisant des modélisations au simple utilisateur qui consulte des données géographique, le SIG peut s'adresser à une gamme très variée d'utilisateur.

Données vecteur, données raster

Les données sont à la base de tout SIG. Elles sont organisées sous forme de couches dites thématiques qui regroupent l'ensemble des informations liées à un sujet spécifique (couche de routes, couche de forêts, couche d'adresses, etc.).

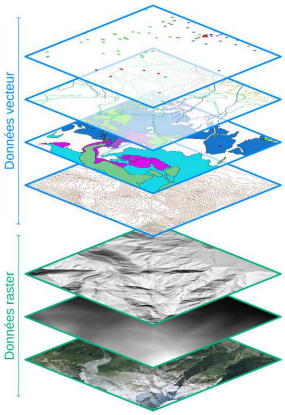
Il existe deux grands types de données géographiques :

Les données vecteur

La représentation géographique de ces objets est constituée de... vecteurs. Ce sont des points, des polylignes (séries de points) ou des polygones (séries de points dont le dernier coïncide avec le premier). À chacun de ces objets sont associées des données alphanumériques permettant de décrire l'objet (par exemple un nom, une catégorie, une spécificité).

Les données raster

Ce sont des matrices ordonnées de points identifiés par leurs coordonnées et associés à des informations comme par exemple l'altitude. Les orthophotographies sont un excellent exemple de données raster, chaque point étant représenté par un pixel.



Un troisième type de données est largement utilisé au sein des SIG, il s'agit des données dites *alphanumérique*. Elles sont, en général, liées à d'autres données géolocalisées par le biais d'un attribut commun (comme un code INSEE ou un nom de commune par exemple) pour les utiliser.

Méthodes de stockage

Deux méthodes sont principalement utilisées pour le stockage des données : les fichiers plats et les bases de données.

Les fichiers plats ont longtemps été utilisés, propulsés par les principaux éditeurs de logiciels SIG. Voici quelques-uns des formats les plus couramment rencontrés.

Données vecteur :

- DGN : DesiGN, format natif du logiciel MicroStation de Bentley Systems, principal concurrent du format `.dwg` d'Autodesk.
- DXF : Drawing eXchange Format, format d'échange de données créé par Autodesk qui permet le transfert de données vecteur.
- DWG : DraWinG, format natif du logiciel AutoCAD d'Autodesk, principal concurrent du format `.dgn` de Bentley Systems.
- GeoJSON : encodage des données utilisant la norme JSON (JavaScript Object Notation).
- GPX : GPS eXchange Format, format d'échange de données GPS contenant des points, des traces ou des itinéraires.
- KML : Keyhole Markup Langage, stockage des données avec le formalisme XML.
- Mapinfo : développé par Pitney Bowes Software pour le logiciel MapInfo et constitué d'un ensemble de fichiers `.tab`, `.dat`, `.id` et `.map`.
- MIF/MID : MapInfo Interchange Format, format d'échange développé par Pitney Bowes Software contenant deux fichiers : un `.mif` (contenant les données et les géométries) et un `.mid` (informations attributaires).
- Shapefile : développé par ESRI et principal fichier de données vecteur SIG, il est constitué d'un ensemble de fichiers d'extensions dont au moins les trois suivants : un `.shp` (fichier de forme), un `.shx` (index géométrique) et un `.dbf` (données attributaires).

Données raster :

- ECW : Enhanced Compression Wavelet, format de fichier compressé avec perte.
- GeoTIFF : fichier TIFF associé à des informations de géoréférencement.
- JPEG2000 : fichier compressé.
- MrSID : Multi-Resolution Seamless Image Database.

Les bases de données sont maintenant à la base de toute infrastructure SIG moderne. Elles reposent sur des logiciels permettant de stocker les données au sein de tables organisées entre elles. PostgreSQL, Oracle, MySQL, SQL Server sont des systèmes de gestion de bases de données pouvant gérer l'aspect géographique.

Il existe également des formats hybrides, comme par exemple SpatiaLite ou GeoPackage. Ils permettent de stocker les données dans des tables regroupées au sein de fichiers, alliant ainsi les avantages des bases de données à la portabilité des fichiers plats.

Géolocalisation et systèmes de projection

Les données spatiales sont par définition géolocalisées, c'est-à-dire qu'elles sont associées à des coordonnées (valeurs numériques) permettant de les localiser à la surface du globe terrestre. Problème, la Terre possède une surface complexe, il s'agit d'une pseudo sphère aplatie aux pôles et qui subit des déformations constantes (mouvement des plaques, marées terrestres et océaniques) et dont le centre de gravité varie en raison du déplacement des masses atmosphériques, océaniques et terrestres. Définir un système de coordonnées est donc relativement complexe mais comprendre son fonctionnement est simple.

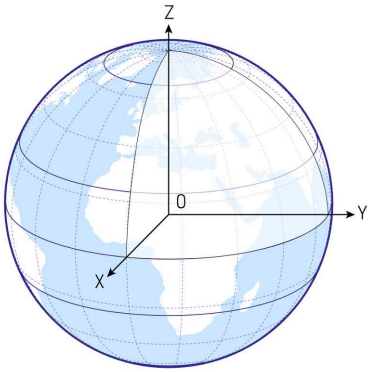
Attention *› Plusieurs termes peuvent coexister pour une même définition ce qui peut compliquer la recherche d'informations sur le sujet.*

Système de référence terrestre (SRT)

Il s'agit d'un repère tridimensionnel OXYZ dont :

- le centre O est positionné le plus proche possible du centre de gravité terrestre ;
- l'axe OZ est positionné le plus proche possible de l'axe de rotation de la Terre ;
- l'axe OX est dans le plan équatorial (90° de l'axe OZ) et aligné avec le méridien défini comme origine ;
- l'axe OY est dans le plan équatorial (90° de l'axe OZ) et à 90° de l'axe OX.

Figure 1.1 : *Système de Référence Terrestre (SRT)*



Le système de référence terrestre présenté ici prend Greenwich comme méridien d'origine.

Note › *Un SRT peut également être appelé* système de coordonnées géodésiques *ou* système de coordonnées géocentrique.

Un SRT peut être qualifié de :

- global, lorsqu'il est déterminé pour le monde entier. Les coordonnées dans un tel système évoluent dans le temps en raison de la dérive des continents.
- local, lorsqu'il est déterminé pour une région donnée. Le but est d'éviter d'avoir une variation des coordonnées dans le temps De plus, une précision accrue est obtenue sur la zone concernée.

Système de coordonnées géographiques

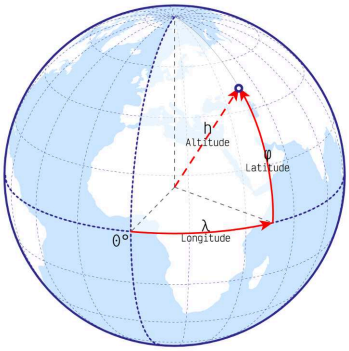
L'utilisation d'un SRT avec des coordonnées X,Y,Z est en général réservé aux calculs GNSS (GALILEO, GPS, GLONASS, BEIDOU, etc.). Il est plus fréquent d'utiliser les SRT en association avec un ellipsoïde de révolution dont le demi-grand axe est dans le plan de l'équateur et le demi-petit axe confondu avec l'axe OZ. Pour faire simple, cet ellipsoïde est en fait une approximation du geoïde terrestre qui est lui-même une modélisation de l'altitude 0.

$$ellipsoïde \simeq géoïde \simeq modélisation\ de\ l'altitude\ 0$$

Lorsqu'un SRT est associé à un ellipsoïde, on peut alors parler de *système de coordonnées géographiques* ou de *datum* (même si le terme de SRT est toujour valide) et il devient alors possible de définir les coordonnées d'un point M sur cette surface grâce à :

- la longitude : angle entre le méridien d'origine et le méridien du point M ;
- la latitude : angle entre le plan de l'équateur et la normale à l'ellipsoïde au niveau du point M ;
- l'altitude : hauteur entre la surface de l'ellipsoïde et le point M.

Figure 1.2 : *Système de coodonnées géographiques (latitude/longitude)*



Le système de référence terrestre présenté ici prend Greenwich comme méridien d'origine.

Voici quelques exemples de systèmes de coordonnées géographiques :

- ITRF – International Terrestrial Reference Frame : SRT mondial (et donc global) qui tente de reproduire le plus fidèlement possible le globe terrestre. Aucun ellipsoïde de référence.
- WGS 84 – World Geodetic System 1984 : SRT américain global associé au système GPS. Ellipsoïde de référence : WGS 84.
- NAD 83 – North American Datum of 1983 : SRT nord américain local. Ellipsoïde de référence : GRS 1980.
- ETRS 89 – European Terrestrial Reference System 1989 : SRT européen local. Ellipsoïde de référence : GRS 1980.
- NTF – Nouvelle Triangulation de la France : SRT français périmé. Ellipsoïde de référence : Clarke 1880.
- RGF 93 – Réseau Géodésique Français : SRT français local actuel. Ellipsoïde de référence : GRS 1980.
- CHTRS 95 – Swiss Terrestrial Reference System 1995 : SRT suisse global. Ellipsoïde de référence : GRS 1980

- CH 1903+ – Mise à jour de Suisse 1903 : SRT suisse local. Ellipsoïde de référence : Bessel 1841.

Note *› Les ellipsoïdes WGS 84 et IAG GRS 80 sont presque identiques et ne diffèrent que de 0,1 mm pour leur demi-petit axe.*

Note *› Plusieurs SRT découlent du SRT ITRF. Par exemple, l'ETRS 89 reprend la définition de l'ITRF mais se base sur les stations GPS européennes afin de ne pas subir les effets de la dérive du continent européen. Il s'agit donc d'un SRT local. Il en résulte un décalage grandissant d'environ 2,5 cm par an entre l'ITRF et l'ETRS 89.*

Système de coordonnées projetées

Pour une utilisation cartographique, les coordonnées issues d'un système de référence terrestre, qui sont non planes, doivent subir une transformations afin de pouvoir être représentées sur un plan. Le but est de pouvoir utiliser ces coordonnées dans un repère affine (ou repère cartésien) composé d'un point d'origine et de vecteurs de base (O, X, Y et Z par exemple).

Pour cela, on utilise des équations qui permettent de transformer l'ellipsoïde associé au SRT en une surface plane. Cette transformation est appelée projection des coordonnées.

Les projections sont classées selon leurs propriétés :

- Type de conservation :
  - projection équivalente : conserve les aires.
  - projection conforme : conserve les angles (et donc les formes).
  - projection aphylactique : conserve les distances.
- Type de représentation du globe terrestre :
  - projection cylindrique : projection du globe sur un canevas cylindrique. Le cylindre est généralement d'axe Nord-Sud mais il peut également croiser perpendiculairement l'axe nord-sud, dans ce cas la projection est dite transverse.
  - projection cylindrique : projection du globe sur un canevas cylindrique d'axe nord-sud.
  - projection conique : projection du globe sur un canevas conique dont le sommet est situé sur l'axe Nord-Sud.
  - projection azimutale : projection du globe sur un canevas plan et tangent ou sécant au globe.
- Les autres : des projections uniques qui ne rentrent pas dans les types précédents.

Voici quelques exemples de projections :

- projection cylindrique équidistante
- projection de Mercator : cylindrique conforme
- projection Transverse Universelle de Mercator (UTM) : cylindrique transverse conforme
- projection de Mercator oblique : cylindrique conforme
- projection de Lambert : conique conforme
- projection Equal Earth : pseudo cylindrique équivalente

Un système de coordonnées projetées est ainsi défini par un système de référence terrestre et une projection.

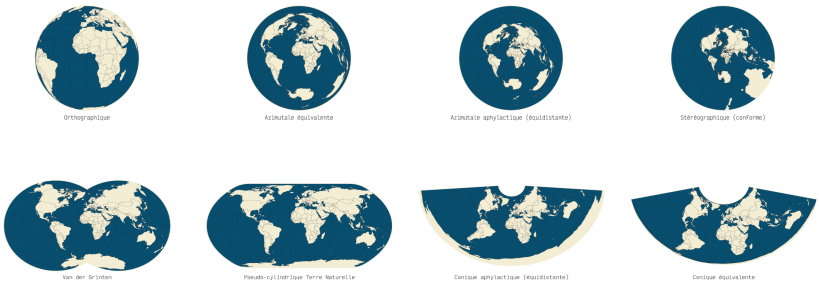
Note *› Un système de coordonnées projetées peut également être appelé "système de coordonnées planes" (en référence au plan) ou "système de coordonnées cartographiques".*

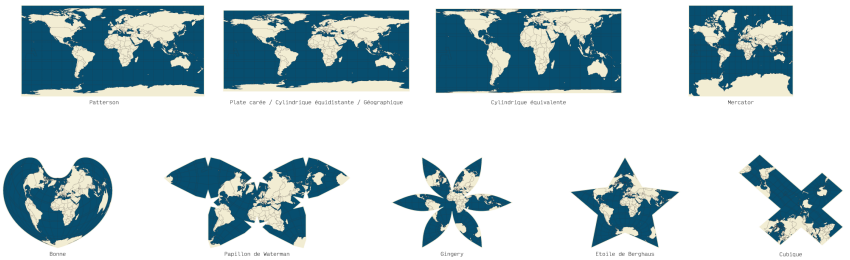
Voici quelques systèmes de coordonnées projetées régulièrement utilisés en France métropolitaine :

- RGF93 – Lambert 93 : projection conique, conforme.
- RGF93 – Lambert 9 zones (CC42 à CC50) : projection conique, conforme.
- NTF – Lambert 4 zones (I à IV) : projection conique, conforme (systèmes de coordonnées dépréciés mais encore rencontrés).
- WGS 84 – Pseudo-Mercator : projection cylindrique, conforme.

Chaque système de coordonnées projetées entraîne une certaine déformation dans l'affichage des données cartographiques.

Figure 1.3 : *Plusieurs systèmes de coordonnées projetées*





la Terre est centrée sur l'équateur au niveau du méridien de Greenwich pour chacune de ces illustrations.

Identificateur de Référence Spatiale – SRID

Afin d'identifier simplement les différents systèmes de coordonnées, chacun d'eux est associé à un code d'identification : le SRID pour Spatial Reference IDentifier. Bien que plusieurs systèmes de codification existent en parallèle, celui créé par l'European Petroleum Survey Group (EPSG) est le plus répandu et le terme de code EPSG est souvent synonyme de SRID (bien qu'étant un abus de langage). Ce groupe est maintenant devenu le Comité de topographie et de positionnement de l'Association internationale des producteurs de pétrole et de gaz (OGP).

Un même système de coordonnées peut recevoir plusieurs SRID selon son utilisation. Voici quelques SRID fréquemment utilisés :

Tableau 1.1 : *Code SRID*

SRID	Système de référence terrestre	Projection	Remarque
4965	RGF 93	-	Coordonnées 3D - Unité : degré
4171	RGF 93	-	Coordonnées 2D - Unité : degré
4937	ETRS 89	-	Coordonnées 3D - Unité : degré
4258	ETRS 89	-	Coordonnées 2D - Unité : degré
4979	WGS 84	-	Coordonnées 3D - Unité : degré
4326	WGS 84	-	Coordonnées 2D - Unité : degré

SRID	**Système de référence terrestre**	**Projection**	**Remarque**
2154	RGF 93	Lambert 93	Unité : mètre
3942	RGF 93	Lambert zone 1 – CC42	Unité : mètre
3943	RGF 93	Lambert zone 2 – CC43	Unité : mètre
3944	RGF 93	Lambert zone 3 – CC44	Unité : mètre
3945	RGF 93	Lambert zone 4 – CC45	Unité : mètre
3946	RGF 93	Lambert zone 5 – CC46	Unité : mètre
3947	RGF 93	Lambert zone 6 – CC47	Unité : mètre
3948	RGF 93	Lambert zone 7 – CC48	Unité : mètre
3949	RGF 93	Lambert zone 8 – CC49	Unité : mètre
3950	RGF 93	Lambert zone 9 – CC50	Unité : mètre
27562	NTF	Lambert centre standard – Zone II	Unité : mètre
27572	NTF	Lambert Zone II carto	Unité : mètre
3857[a]	WGS 84	Pseudo Mercator	Projection utilisée par les principaux services de cartographie web. Unité : mètre
3395	WGS 84	Mercator	Unité : degré
4151	CHTRS 95	-	Unité : degré
4150	CH 1903+	-	Unité : degré
2056	CH 1903+	LV95 (MN95 en français)	Unité : mètre

[a]Le code 900913 est également utilisé mais non officiel.

1.2. PostGIS

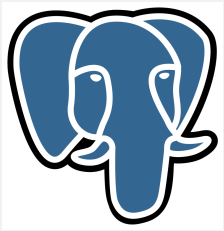
Qu'est-ce que PostGIS ?

Logo de PostGIS reprenant Slonik, l'éléphant de PostgreSQL portant un globe terrestre.

PostGIS est la contraction de "PostgreSQL" et "GIS" (abréviation anglaise pour SIG). Il s'agit d'une extension de PostgreSQL qui lui apporte le statut de base de données spatiales en ajoutant des types de données et des fonctions de traitement spécifiques aux données spatiales.

Note *› Bien que PostGIS ne soit qu'une extension de PostgreSQL, il est courant de la qualifier abusivement de base de données spatiales alors que c'est le couple PostgreSQL + PostGIS qui l'est.*

POSTGRESQL, UN SGBDR OPEN-SOURCE TRÈS ROBUSTE



Slonik, logo de PostgreSQL depuis 2003.

PostgreSQL est un Système de Gestion de Base de Données Relationnel (SGBDR). Pour faire simple, il est possible d'imaginer PostgreSQL comme une sorte de "super" classeur Excel pouvant contenir des milliers de feuilles de façon organisée et proposant le langage SQL pour interroger et interagir avec les données. Robuste, fiable et

très puissant, il est capable de manipuler de très grande quantité de données même dans des situations difficiles avec des fonctionnalités riches et avancées.

PostgreSQL fonctionne sur la plupart des systèmes d'exploitation : Windows, Linux et UNIX et est facile à prendre en main.

Il respecte particulièrement la norme AINSI-SQL 2008 et supporte plus d'une douzaine de langages de programmation : Python, Java, Perl, C/C++, Ruby tout en possédant son propre langage de programmation, le PL/pgSQL, qui ressemble beaucoup au PL/SQL d'Oracle.

Il est open-source et développé par plusieurs milliers de contributeurs à travers le monde issus de nombreuses entreprises, ce qui permet des mises à jour régulières. Cet aspect apporte également une immense communauté sur internet qui sera à même de répondre à toutes les questions.

Éléments de chronologie :

1974 : création d'INGRES (INtelligent Graphic RElational System)

1985 : prototypage de Postgres : Post-INGRES

1995 : Postgres devient Postgre95 avec l'ajout du support du SQL

1996 : Postgres95 devient PostgreSQL

Créateur : Michael Stonebraker, université de Californie, département des sciences informatiques de Berkeley.

Plateformes : Solaris, SunOS, macOS X, HP-UX, AIX, Linux, IRIX, Digital Unix, BSD, NetBSD, FreeBSD, OpenBSD, SCO Unix, NeXTSTEP, UnixWare et toutes sortes d'Unix ainsi que Windows.

Limites :

- Taille maximum d'une BDD : illimitée
- Taille maximum d'une table : 32 To
- Taille maximum d'un enregistrement : 1,6 To
- Taille maximum d'un attribut par enregistrement : 1 Go
- Nombre maximum d'enregistrements : illimité
- Nombre maximum d'attributs par table : 250 à 1600 selon le type de colonne
- Nombre maximum d'index par table : illimité

Exemples d'entreprises utilisant PostgreSQL : IGN, Météo France, Le Bon Coin, Skype, LastFM, McAfee, Trend Micro, l'autorité de l'aviation fédérale (FAA) américaine ou encore l'application de course à pied RunKeeper ; le domaine .org est aussi géré avec PostgreSQL.

La première version de PostGIS (0.1) a été lancée le 31 mai 2001 par Refractions Research Inc. (société de conseil en base de données et en SIG située au Canada) dans le but de pouvoir stocker des objets spatiaux de type vecteur, l'analyse spatiale n'était pas encore au programme. Son développement est maintenant coordonné par la OSGeo Foundation (Fondation open-source geospatiale), une organisation non gouvernementale fondée en 2006 pour soutenir et construire une offre de logiciels open-source en géomatique. Un grand nombre d'organisations et de développeurs participent à son développement.

Un gros travail a été mené sur le stockage des géométries en réduisant l'espace utilisé ainsi qu'en maximisant la compatibilité avec la norme ISO SQL-Multimedia (SQL/MM) qui a défini le mode de représentation des objets du monde réel.

Petit à petit, le nombre de fonctions a augmenté et s'est structuré grâce à la spécification *Simple Feature for SQL* (SFSQL) de l'Open Geospatial Consortium (OGC) qui donne des indications pour le nommage des fonctions et les prérequis. La capacité de traitement des fonctions a été grandement améliorée lorsqu'elles ont commencé à se baser sur un second projet de l'OSGeo Foundation : GEOS pour *Geometry Engine, Open Source*. Il s'agit d'une bibliothèque qui fournit un ensemble d'algorithmes de traitements spatiaux respectant la spécification SFSQL.

D'autres projets de l'OSGeo Foundation sont utilisés au sein de PostGIS :

- GDAL (Geospatial Data Abstraction Library), une bibliothèque utilisée pour lire et interagir avec différents formats de données ;
- PROJ, une bibliothèque contenant des outils permettant d'effectuer des transformations de coordonnées.

Pourquoi PostGIS ?

Le monde des SIG est vaste, aussi bien en matière de type de stockage des données que de logiciel d'édition de ces dernières. Alors pourquoi utiliser PostGIS plutôt qu'un autre logiciel ?

Lorsque l'on travaille avec des données SIG, de nombreux fichiers sont utilisés. Au fur et à mesure que le système grandit, le nombre de données augmente en même temps

que le nombre de fichiers. Il devient alors compliqué de les gérer et de les organiser. Souvent, plusieurs types de fichiers cohabitent et certains impliquent l'utilisation de logiciels spécifiques pour pouvoir les utiliser. La meilleure des solutions est alors de centraliser toutes ces données au sein d'une base de données. L'accès est plus rapide, mieux organisé, plus sécurisé et surtout compatible avec la plupart des logiciels SIG.

PostGIS repose sur le système de gestion de base de données PostgreSQL. Il bénéficie donc de tous les avantages de ce dernier : facilité d'installation et de gestion, puissance de traitement, respect de la norme SQL, compatibilité avec tous les systèmes d'exploitation, mises à jour régulières, développement open-source. PostgreSQL est en plus totalement gratuit, ce qui est un excellent point de démarrage pour créer une première base de données SIG.

PostGIS est lui aussi gratuit. Son développement est basé sur les standards de l'Open Geospatial Consortium (OGC) et les mises à jour régulières améliorent constamment ses performances. L'installation est simple, de même que les mises à jour.

Un avantage de PostGIS vient du fait qu'il s'agit à la fois d'un moyen de stockage de données et d'un outil d'analyse spatiale. Il est ainsi possible de faire calculer des opérations par la base de données sans logiciel tiers. De plus, il supporte à la fois les données vecteur et les données raster ainsi que de nombreux formats comme le GeoJSON, le KML, les MVT (Mapbox Vector Tiles) ce qui permet d'interagir avec de multiples applications web.

Avec PostGIS, il devient possible de traiter efficacement de grandes quantités de données spatiales, simplement et beaucoup plus rapidement qu'à partir de fichiers plats. Il est également aisé d'automatiser les traitements pour des tâches régulières.

Une base de données simplifie le travail à plusieurs. En effet, il est tout à fait possible d'intervenir en même temps sur les mêmes tables, PostgreSQL s'occupe de conserver l'intégrité des données.

Il existe plusieurs sociétés qui proposent une assistance sur PostGIS mais son taux d'adoption est tel qu'il en résulte une communauté d'utilisateurs à même de répondre à n'importe quelle question.

Pour toutes ces raisons, PostgreSQL + PostGIS est aujourd'hui le système de gestion de bases de données spatiales le plus utilisé dans le monde des SIG. Voici quelques organisations qui l'ont choisi pour gérer leurs données :

- OpenStreetMap (OSM), projet collaboratif de cartographie en ligne.
- l'Institut Géographique National (IGN) (Frane);

- CartoDB, service de cartographie en ligne ;
- la National Oceanographic and Atmospheric Administration (NOAA – US) ;
- Instagram (qui a choisi PostgreSQL en raison de PostGIS).

Alternatives à PostGIS

PostgreSQL + PostGIS n'est bien sûr pas la seule base de données spatiales qui existe. Plusieurs alternatives se côtoient, qu'elles soient propriétaires ou ultra-légères, anciennes ou toutes récentes.

Tout a commencé avec Oracle Spatial Data Option (SDO) (Oracle 7), renommée ensuite Oracle Spatial. Il s'agissait à l'origine d'une extension onéreuse, c'est la raison pour laquelle une version basique offrant le strict minimum en terme de support spatial (Oracle Locator) était incluse dans chaque installation. Cette extension est maintenant incluse gratuitement avec la licence Oracle. Les possibilités offertes sont excellentes tant sur le plan performances que fonctionnalités, même si elles n'atteignent pas le niveau de PostGIS.

MySQL et MariaDB sont beaucoup utilisés pour le développement web et supportent les données spatiales. Cependant, la maturité de leurs moteurs SQL et de leurs moteurs d'indexation ne peuvent rivaliser avec PostGIS malgré l'amélioration constante de leurs performances. Les analyses spatiales poussées sur de gros jeux de données sont donc à exclure dans ce cas.

Microsoft SQL Server a introduit le type spatial avec SQL Server 2008. Les fonctionnalités spatiales restent très limitées en comparaison de PostGIS.

SpatiaLite et GeoPackage sont les plus récents et reposent tous les deux sur SQLite qui est une base de données portable open-source. SpatiaLite se base sur les mêmes bibliothèques que PostGIS (GEOS, PROJ et GDAL), ce qui lui permet d'utiliser les mêmes types géométriques et le même système de nommage de fonctions. Il gère également le format raster. GeoPackage, lui, est avant tout dédié au stockage des données vecteur et raster et n'inclut pas de fonctions d'analyse spatiale. Il est de plus en plus utilisé, notamment par QGIS comme format d'export par défaut. Cependant, ses possibilités de requêtage sont limitées notamment en terme d'agrégation mais le fait que les données soient stockées dans un fichier unique le rend facilement transportable et déployable, y compris sur des terminaux mobiles.

Les bases NoSQL (Azure Cosmo DB, Google Cloud Firestore, MongoDB, Elasticsearch) implémentent également le support des données spatiales même si ces fonctionnalités sont bien loin des possibilités offertes par PostGIS.

Bien qu'il existe plusieurs alternatives, PostGIS conserve d'excellents atouts sur ses adversaires : sa gratuité certes mais aussi et surtout ses performances parmi les meilleures.

1.3. PostgreSQL dans la pratique

PostgreSQL est facile à utiliser.

Architecture Client/Server

PostgreSQL est un serveur, autrement dit un logiciel qui offre des services : ici la consultation d'une base de données. Pour utiliser ces services, il faut recourir à un logiciel client. Il s'agit du même principe de fonctionnement qu'internet :

- un serveur offre un service : le world wide web (www) ;
- un client permet sa consultation : le navigateur.

Le logiciel client va ainsi envoyer des requêtes au serveur PostgreSQL ; ce dernier les *écoute* puis retourne sa réponse.

Note *› Le terme de serveur peut à la fois qualifier une machine physique et un logiciel. Un serveur est une entité qui peut être contactée et qui doit répondre à ce contact.*

Comme lorsque l'on travaille avec des fichiers, il faut établir une connexion avec PostgreSQL pour accéder aux données. Avec un fichier, il suffit d'un chemin d'accès, alors que plusieurs paramètres sont nécessaires pour se connecter à une base de données :

- l'hôte du serveur (l'adresse de l'immeuble en quelque sorte) qui est souvent l'adresse IP de la machine où est installé PostgreSQL ;
- le numéro de port (le n° d'appartement dans notre analogie) qui est par défaut le 5432 (mais il n'est pas rare de voir des installations faites sur le port 5433, 5434, etc.) ;

Avec un tel fonctionnement, plusieurs instances de PostgreSQL peuvent être installées et tourner en parallèle sur une même machine. Ces instances auront des numéros de ports différents.

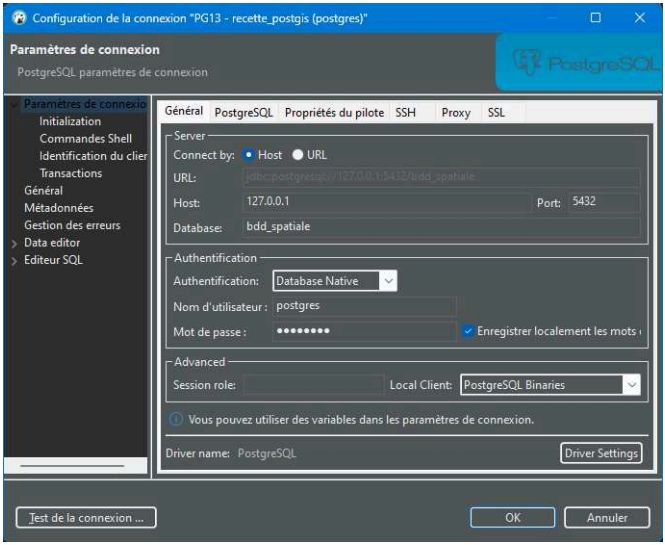
La connexion s'établit ensuite au moyen d'un login (identifiant) et d'un mot de passe donnant accès aux différents éléments de la base de données conformément aux règles de droits qui auront été définies.

Parmi le nombreux clients qui existent, voici les principaux :

- `psql` : client en ligne de commande fourni par défaut avec PostgreSQL ;

- `pgAdmin IV` : client graphique fourni par défaut avec PostgreSQL qui permet la visualisation des données cartographiques ;
- `DBeaver` : client graphique gratuit qui permet de se connecter à de nombreux types de bases de données dont PostgreSQL. Il permet la visualisation des données cartographiques ;
- `QGIS` : logiciel SIG, il contient également un client permettant l'accès à PostgreSQL.

Figure 1.4 : *Interface de paramétrage d'une connexion à PostgreSQL avec DBeaver*



Astuce › *Bien que pgAdmin soit beaucoup utilisé, nous lui avons préféré DBeaver pour réaliser les exemples dans ce livre. Outre son développement open-source et sa gratuité (comme pgAdmin), il s'agit d'un formidable client pour PostgreSQL qui dispose de nombreuses options plus avancées que celles de pgAdmin et bien plus pratiques à l'utilisation.*

Sans entrer dans les détails de fonctionnement de ces clients, voici les concepts de base.

L'interface du logiciel client permet de visualiser les différents objets présents dans la base ainsi que leurs propriétés (et leurs données dans le cas des tables). Un éditeur incorporé permet de rédiger des lignes de code en langage SQL (les requêtes), qui seront exécutées par le serveur. Les résultats calculés celui-ci sont envoyés au logiciel client qui les affiche dans une fenêtre dédiée.

Note › *Toutes les requêtes SQL indiquées dans cet ouvrage sont à exécuter via l'éditeur de votre logiciel client qui vous permettra également d'en visualiser les résultats.*

PostGIS étant une extension de PostgreSQL, pour bénéficier de ses fonctionnalités et de sa puissance de calcul, il est nécessaire d'utiliser le langage de PostgreSQL (le SQL). Il est donc important de connaître la structuration d'une base de données et d'avoir quelques bases en requêtage SQL.

Structuration de la base de données

Au sein de PostgreSQL, les données sont structurées, ce qui permet de les retrouver facilement. Voici les regroupements de données que l'on trouve, chaque élément étant un regroupement du précédent :

- le champ ;
- le tuple (ligne d'une table) ;
- la relation (une table) ;
- le schéma ;
- la base de donnée.

Note *› Les tables peuvent être regroupées au sein de schémas qui sont des sortes de répertoires n'existant qu'à un seul niveau. Il n'est donc pas possible de créer un schéma dans un schéma.*

Note *› Le schéma est l'objet dans lequel tout objet d'une base de donnée est stocké, que ce soit une table, une fonction ou autre.*

Par défaut, chaque base de données est créée avec un schéma nommé *public* qui est le schéma par défaut. Il contient les fonctions de base et c'est dans ce schéma que sont installées par défaut chaque extension. Il est cependant parfois recommandé d'installer certaines extensions hors du schéma public, dans leur propre schéma. Cela facilite la gestion de celles-ci.

Pour interroger des données au sein de la base, il faut spécifier le nom de la table ainsi que le nom du schéma dans lequel elle se trouve. Ceci permet d'avoir plusieurs tables portant le même nom mais dans des schémas différents.

Les rôles sont l'équivalent des utilisateurs dans PostgreSQL. Il s'agit d'un concept un peu particulier car un rôle est à la fois l'équivalent d'un utilisateur et l'équivalent d'un groupe d'utilisateurs. La connexion au serveur se fait donc par le biais d'un rôle (concept d'utilisateur) qui peut lui-même appartenir à un ou plusieurs autres rôles (concept de groupe d'utilisateurs).

Les rôles permettent de définir des droits d'accès aux différents objets de la base. Avant la version 15 de PostgreSQL, le schéma public était accessible par défaut à tout rôle

d'où l'utilisation de celui-ci comme schéma par défaut pour l'installation de toute nouvelle extension afin de la rendre directement accessible à n'importe quel rôle. À partir de la version 15, il est nécessaire de gérer les droits d'accès au schéma public pour chaque rôle, ce qui rend l'utilisation de ce schéma tout aussi contraignant qu'un schéma spécifique à chaque extension.

Lorsqu'une table est interrogée et que son schéma n'est spécifié, le système recherche cette table dans une liste de schémas préétablie. Cette liste est stockée dans le chemin de recherche (*search_path* en anglais). Il s'agit d'une variable (qui peut donc être modifiée) et qui contient par défaut deux schémas : le premier portant le même nom que le rôle utilisé et le second étant le schéma `public`.

Note *› Même si aucun schéma portant le même nom que le rôle utilisé n'existe, celui-ci est toujours défini comme premier schéma de recherche par défaut.*

La requête suivante permet de visualiser le contenu du chemin de recherche :

```
SHOW search_path;
```

Pour ne pas avoir à spécifier le schéma dans lequel vous travaillez, vous pouvez redéfinir le chemin de recherche à l'aide de la requête suivante :

```
SET search_path TO mon_schema,public;
```

Notions de base en SQL

Le SQL est un langage simple qui permet d'interroger les données d'une base PostgreSQL et d'interagir avec. Voici quelques notions de base.

Une requête se termine toujours par un point-virgule `;` celui-ci indique à PostgreSQL que l'instruction est terminée.

Pour le nom des objets (schéma, table, colonne), la norme SQL impose l'utilisation des caractère suivants :

- les 26 lettres de l'alphabet ;
- les dix chiffres de 0 à 9 ;
- le caractère souligné _.

PostgreSQL est plus laxiste et autorise l'utilisation d'accents, d'espace et de tout autre caractère. Il est cependant recommandé de n'utiliser que les caractères de la norme SQL. En effet, certains logiciels sont encore incompatibles avec ceux-ci.

Lorsque qu'un nom d'objet est spécifié, il doit obligatoirement être entouré de guillemets s'il contient d'autres caractères que ceux de la norme SQL. PostgreSQL est sensible à la casse : les noms d'objets contenant des majuscules doivent également être encadrés de guillemets, à défaut de quoi ils seront considérés comme étant en minuscules.

Concernant le nom des commandes (`SELECT` par exemple), PostgreSQL n'est en revanche pas sensible à la casse. Il est souvent plus lisible de les rédiger en majuscules.

L'indentation n'est pas obligatoire mais permet grandement de simplifier la lecture des requêtes.

Les sauts de ligne sont ignorés par PostgreSQL. N'hésitez donc pas à aérer vos requêtes pour qu'elles soient plus lisibles.

Les valeurs textuelles sont entre guillemets simples (`'`), les valeurs numériques n'en ont pas besoin.

Les noms d'objets ne sont pas limités. Il ne faut donc pas hésiter à utiliser des termes entiers plutôt que des abréviations ni à ajouter des préfixes pour mieux identifier les objets.

Chaque objet peut être commenté, ce qui permet de simplifier l'utilisation de la base et de détailler l'organisation de celle-ci. Cela vous aidera particulièrement lorsque vous n'aurez pas mis le nez dans votre base depuis longtemps.

Les requêtes peuvent également contenir des commentaires :

- monolignes (ils débutent alors par un double tiret `--`) : tout ce qui suit un double tiret est ignoré et n'est pas interprété par PostgreSQL. Les commentaires peuvent commencer en fin de ligne à la suite d'une instruction ;
- multilignes (ils débutent par `/*` et finissent par `*/`) : tout ce qui est encadré par ces balises n'est pas interprété par PostgreSQL.

`NULL` n'est pas une valeur, c'est l'absence de valeur. Ainsi, il n'est pas possible de rechercher `ma_valeur = NULL` car il n'y a, par définition, aucun élément à comparer ; il faudra plutôt utiliser `ma_valeur IS NULL`.

Requêtes SQL de base

Dans PostgreSQL, tout est requête : structuration de la base, gestion des droits, création, modification, lecture et analyse des données, etc. Les requêtes SQL sont composées de différents mots clés, suivis d'expressions. Voici plusieurs exemples pour mieux appréhender leur fonctionnement.

Gérer la structure

Les requêtes permettent de gérer les objets de la base et leur structure.

Exemple 1.1 : *Créer une base de données*

La connexion initiale à un serveur PostgreSQL se fait sur la base fournie par défaut et nommée `postgres`. L'idéal est, en règle générale, de travailler dans une base de donnée dédiée.

Commençons donc par créer une base de données dans laquelle nous réaliserons nos exemples. Nous précisons l'encodage des données : l'UTF-8 (UCS Transformation Format 8) qui (pour faire simple) est un encodage supportant les caractères de tous les langages du monde.

```
CREATE DATABASE bdd_spatiale
  ENCODING UTF8
;
```

Pour la suite des exemples, il faut vous connecter à la base de données nouvellement créée.

Exemple 1.2 : *Créer un schéma*

Le schéma pourra contenir des tables, des fonctions et tout un tas d'autres objets.

```
CREATE SCHEMA exemple_notion_sql
;
```

Exemple 1.3 : *Créer une table*

Créer une table simple avec quelques colonnes.

```
-- La table est appelée par son nom
-- qui peut être préfixé du schéma contenant la table
CREATE TABLE exemple_notion_sql.table_simple (
  -- Chaque colonne est définie par son nom et un type de données
  colonne_1 text,
  colonne_2 integer,
  colonne_3 numeric,
  colonne_4 date,
  colonne_5 boolean
)
;
```

Exemple 1.4 : *Modifier une table*

Modifier une table et sa structure.

```
-- Commençons par créer une table de test
CREATE TABLE exemple_notion_sql.table_de_test (
  colonne_1 text
)
;

-- Renommer une table
ALTER TABLE exemple_notion_sql.table_de_test
  RENAME TO nouvelle_table_de_test
;

-- Ajouter une colonne
ALTER TABLE exemple_notion_sql.nouvelle_table_de_test
  ADD COLUMN colonne_2 integer
;

-- Renommer une colonne
ALTER TABLE exemple_notion_sql.nouvelle_table_de_test
  RENAME COLUMN colonne_2 TO colonne_3
;

-- Supprimer une colonne
ALTER TABLE exemple_notion_sql.nouvelle_table_de_test
  DROP COLUMN colonne_3
;
```

Exemple 1.5 : *Supprimer une table*

Supprimer une table et toutes ses données.

```
DROP TABLE exemple_notion_sql.nouvelle_table_de_test
;
```

Éditer des données

Exemple 1.6 : *Insérer des données – Une ligne*

Ajouter des données dans une table simple avec quelques colonnes.

```
INSERT INTO exemple_notion_sql.table_simple (
  -- Chaque colonne dans laquelle des données vont être insérées est appelée
  colonne_1,
  colonne_2,
  colonne_3
)
-- Requête contenant les données dans l'ordre des colonnes appelées
SELECT
  'mon texte',
  2,
  3.14
;
```

Exemple 1.7 : *Insérer des données – Plusieurs lignes*

Ajouter des données dans une table simple avec quelques colonnes.

```
INSERT INTO exemple_notion_sql.table_simple (
  colonne_1,
  colonne_2,
  colonne_3
)
VALUES
  -- Chaque ligne à insérer contient les données dans l'ordre des colonnes
 appelées
  ('Autre texte', 1, 4.12),
  ('Sympa texte', 1, NULL),
  ('Mini texte', 2, NULL),
  ('Texte', 2, NULL),
  ('TEXTE', 1, NULL),
  ('TeXTe', 3, NULL),
  ('tExtE', 3, NULL),
  ('Bim', 3, NULL),
  ('Bam', 2, NULL),
  ('Boum', 3, NULL),
  ('Un', 1, NULL),
  ('Deux', 5, NULL),
  ('Trois', 8, NULL),
  ('Quatre', 4, NULL),
  ('Cinq', 7, NULL),
  ('Six', 2, NULL),
  ('Sept', 6, NULL)
;
```

Exemple 1.8 : *Mettre à jour des données*

Remplacer toutes les valeurs d'une colonne par une autre.

```
UPDATE exemple_notion_sql.table_simple
  -- Définition de la nouvelle valeur pour une colonne spécifique
  SET colonne_3 = 3.14
;
```

Remplacer une (ou plusieurs) valeur(s) d'une colonne.

```
UPDATE exemple_notion_sql.table_simple
  SET colonne_1 = 'Petit texte'
  -- Filtrage des lignes devant être mises à jour
  WHERE colonne_2 = 1
;
```

Exemple 1.9 : *Supprimer des données*

Supprimer certaines lignes spécifiques.

```
DELETE FROM exemple_notion_sql.table_simple
  WHERE colonne_1 = 'mon texte'
;
```

Supprimer toutes les lignes d'une table. (Si vous utilisez la commande suivante, pensez à réinsérer les lignes pour les exemples suivants.)

```
DELETE FROM exemple_notion_sql.table_simple
;
```

Une autre manière de vider une table. (Si vous utilisez la commande suivante, pensez à réinsérer les lignes pour les exemples suivants.)

```
TRUNCATE exemple_notion_sql.table_simple
;
```

Note *› La principale différence entre* DELETE *et* TRUNCATE *réside dans le fait que* TRUNCATE *réalise toute une série d'actions supplémentaires en plus de supprimer toutes les lignes de la table (réinitialisation des index,* VACUUM, ANALYZE, *etc.).*

Afficher des données

Voici les principaux mots clés utilisés pour visualiser des données.

- **SELECT**, suivi des éléments à visualiser [valeurs textuelles ou numériques, colonnes d'une table, fonction permettant de traiter les données] (le résultat sera affiché) ;
- **FROM**, suivi de la ou des tables à partir desquelles visualiser les données ;
- **WHERE**, suivi d'une expression pour filtrer les données ;
- **GROUP BY**, suivi d'une expression pour grouper les données.
- **ORDER BY**, suivi d'une expression pour ordonner les données ;
- **LIMIT** suivi d'une valeur indiquant le nombre de ligne maximum à récupérer (à afficher).

Exemple 1.10 : *Afficher du texte et des nombres*

Une requête simple pour afficher des données.

```
SELECT
  -- Le texte est encadré par des apostrophes
  'Hello world',
  -- Les valeurs numériques sont écrites directement
  2,
  -- Chaque colonne peut être explicitement nommée
  'Mon super test' as "ma colonne 4",
  -- Les expressions mathématiques sont interprétées
  2 + 2 as "Mon calcul",
  -- Les fonctions sont exécutées et le résultat affiché
  lower('ToUt eN MInuScULe')
;
```

```
column1    |column2|ma colonne 4  |Mon calcul|lower           |
-----------+-------+-------------+----------+----------------+
Hello world|      2|Mon super test|         4|tout en minuscule|
```

Exemple 1.11 : *Afficher des données issues d'une table*

Une requête simple pour afficher les données d'une table.

```
SELECT
  -- La colonne est appelée par son nom
  colonne_1,
  -- Les majuscules ne sont pas prises en compte,
  -- cet exemple équivaut à colonne_2
```

```
  ColONne_2,
  -- Pour prendre en compte les majuscules il faut utiliser des guillemets
  "Ma coLOnNe 4",
  -- Une colonne peut être renommée dans les résultats
  colonne_3 as colonne_5,
  -- Les fonctions sont exécutées sur les données de la colonne
  lower(colonne_1)
FROM
-- La table est appelée par son nom
-- qui peut être préfixé du schéma contenant la table
  exemple_notion_sql.table_simple
;
```

Attention, une erreur sera ici remontée car la colonne `Ma coLOnNe 4` n'existe pas, seule `colonne_4` l'est. Vous pouvez commenter la ligne (avec `--`) contenant la colonne `Ma coLOnNe 4` et vous obtiendrez les résultats suivants (tronqués) :

```
colonne_1   | colonne_2 | colonne_5 | lower       |
------------+-----------+-----------+-------------+
Mini texte  |         2 |      3.14 | mini texte  |
Texte       |         2 |      3.14 | texte       |
TeXTe       |         3 |      3.14 | texte       |
tExtE       |         3 |      3.14 | texte       |
Bim         |         3 |      3.14 | bim         |
 ...
```

Exemple 1.12 : *Filtrer les données*

Une requête pour filtrer les données remontées.

```
SELECT
  colonne_2
FROM
  exemple_notion_sql.table_simple
WHERE
  -- L'expression doit renvoyer une valeur boléenne (vrai/faux)
  colonne_2 < 3
;
```

```
 colonne_2 |
-----------+
         2 |
         2 |
         2 |
         2 |
         1 |
         1 |
         1 |
         1 |
```

Exemple 1.13 : *Grouper les données*

Un requête pour grouper des données et visualiser les valeurs différentes qui existent.

```
SELECT
  colonne_2
FROM
  exemple_notion_sql.table_simple
GROUP BY
  colonne_2
;
```

```
 colonne_2 |
-----------+
         3 |
         5 |
         4 |
         6 |
         2 |
         7 |
         1 |
         8 |
```

Exemple 1.14 : *Ordonner les données*

Une requête pour ordonner les données visualisées.

```
SELECT
  colonne_2
FROM
  exemple_notion_sql.table_simple
ORDER BY
  colonne_2
;
```

```
colonne_2 |
----------+
        1 |
        1 |
        1 |
        1 |
        2 |
        2 |
        2 |
        2 |
        3 |
        3 |
        3 |
        3 |
```

```
         4 |
         5 |
         6 |
         7 |
         8 |
```

Exemple 1.15 : *Limiter les données*

Une requête limitant le nombre de résultats remontées.

```
SELECT
  colonne_2
FROM
  exemple_notion_sql.table_simple
LIMIT 3
;
```

```
 colonne_2 |
-----------+
         2 |
         2 |
         3 |
```

Exemple 1.16 : *Combiner les mots clés*

Une requête combinant les différents mots clés vus précédemment.

```
SELECT
  colonne_2
FROM
  exemple_notion_sql.table_simple
WHERE
  colonne_2 < 5
GROUP BY
  colonne_2
ORDER BY
  colonne_2
LIMIT 3
;
```

```
 colonne_2 |
-----------+
         1 |
         2 |
         3 |
```