Fiches-outils : Extraction et manipulation de données OpenStreetMap

Matthieu Viry

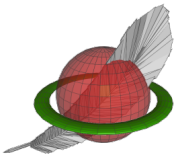
Fiches-outils :
Extraction et manipulation de données OpenStreetMap

Table des matières

Aout 2015 – Matthieu Viry – UMS RIATE
Fiches réalisées dans le cadre d'une étude commanditée par le Commissariat Général à l'Égalité des Territoires.

SQLite / SpatiaLite

Statut : SQLite : Domaine public / *SpatiaLite* : Mozilla Public License
Sites web de référence : www.sqlite.org / www.gaia-gis.it/gaia-sins/
Développé/maintenu par : Voir les 2 sites web de référence
Version actuelle : SQLite : 3.8.10.2 / *SpatiaLite* : 4.2.0



Présentation

SQLite est une solution de gestion de bases données relationnelles, contenues dans une bibliothèque logicielle écrite en C, implémentant une majeure partie des spécifications du langage SQL[1] et des propriétés *ACID*. Contrairement à d'autres SGBDR, celui ci ne repose pas sur le concept de "client-serveur"[2]. L'accès à la base de données se fait par l'ouverture du fichier binaire, indépendant de la plate-forme, contenant les différentes tables, index, action différées, etc. Cette alternative permet par exemple d'éviter la multiplication de tables stockées au format texte (txt ou csv par exemple) par certaines applications, mais elle connaît aussi ses limites[3] lors d'un usage intensif où de nombreux clients pourraient être amenés à utiliser la base de données. De plus on notera que *SQLite* n'implémente pas de gestion des droits à proprement parlé mais utilise celle de la plate-forme sur laquelle la base de données est utilisée.

SpatiaLite est une extension de *SQLite* (comme *PostGIS* est une extension de *PostgreSQL*). Le chargement de l'extension va permettre la manipulation de données spatiales dans *SQLite*.

SpatiaLite implémente le support des données spatiales selon les spécifications de l'*Open Geospatial Consortium* (OGC), ces spécifications étant souvent appelées sous le terme d'*OpenGIS*[4]. Le modèle de géométrie ainsi qu'une grande partie des fonctions ou traitements clés sont donc similaires à ceux de nombreux logiciels SIG ou des autres SGBD spatiaux. L'implémentation de ces standards permet une bonne interopérabilité entre les différents outils de gestion des données spatiales.

A ce titre *SQLite/SpatiaLite* peut donc être comparée aux autres SGBD spatiaux (tels que *PostgreSQL/PostGIS*, *SQL Server* ou *Oracle Spatial*).
Les points forts de *SQLite/SpatiaLite* sont les suivants :

- pas d'installation lourde ni d'administration particulière à prévoir,
- facilité d'échange des données (un fichier pour contenir l'ensemble de la BD : tables, métadonnées, index, *triggers*, etc.),
- léger à l'utilisation (convient à de nombreux types de machines),
- performances correctes.

On note que l'extension *SpatiaLite* n'est pas à proprement parler nécessaire pour le stockage et la manipulation de données spatiales. *SpatiaLite* rend toutefois possible les traitements indispensables sur les géométries et permet notamment la gestion de la projection et des transformations spatiales.

La solution *SQLite/SpatiaLite* va donc avoir un intérêt dans différents cas, notamment lors de l'échange de données, s'effectuant ici via un unique fichier binaire contenant l'ensemble de la base de données (évitant ainsi une profusion de fichiers et garantissant facilement par exemple un échange des métadonnées avec les données) ou lors du traitement/croisement de données qui nécessiterait d'utiliser plusieurs logiciels tels

1 Dans sa révision SQL-92 : http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt (Consulté le 02/06/2015).
2 L'architecture détaillée de SQLite est décrite à l'adresse https://www.sqlite.org/arch.html (Consulté le 02/06/2015)
3 Les limites de SQLite sont documentés à l'adresse https://www.sqlite.org/limits.html (Consulté le 02/06/2015)
4 http://www.opengeospatial.org/standards/as (Consulté le 02/06/2015)

qu'un *SYLK*, un logiciel SIG, etc. et qui peut ici être réaliser directement en SQL dans la base de données. *SQLite* dispose de différentes extensions permettant une interopérabilité avec différentes sources de données, tel que le module *VirtualPG* (permettant une interaction avec les bases de données *PostGreSQL/PostGIS*).
Enfin l'extension *SpatiaLite* est également enrichie par l'apport de plusieurs outils présents dans le package *spatialite-tools*[5], tels que *OSM-tools*[6] (permettant par exemple d'importer des données OSM, d'y effectuer des calculs de distance/temps de parcours sur un réseau précédemment créé, etc.) , *XML-tools* (permettant de manipuler des données au format XML) ou *SHP_doctor* (permettant de corriger des géométries fournies au format shapefile).

Les utilisations qui en sont faites

Le système de base de données *SQLite* est utilisé par de nombreux appareils mobiles ou différents types d'applications, notamment pour le stockage et l'accès aux fichiers de configurations inhérents au fonctionnement de l'appareil ou du logiciel (c'est par exemple le cas de Skype, de l'outil Zotero, des navigateurs web Opera, Chrome, Mozilla Firefox et Safari mais également de l'*iPhone* pour le le stockage des messages textes). *SQLite* est également utilisé par défaut par des outils tels que le framework Django.

L'outil peut être utilisé en ligne de commande ou bien via un GUI[7] fournit par les développeurs de *SpatiaLite*. Il est également possible d'utiliser un des autres GUI prévu pour *SQLite* (par exemple *DB Browser for SQLite* [3]) ou d'utiliser des interfaces comme celles présentes dans Qgis (via son gestionnaire de base de données ou via le module *QSpatialite* par exemple).
Les requêtes suivent les standards du langage SQL et les fonctions spatiales ont des noms similaires à celles de *PostGIS* (cf. [2] pour la liste des fonctions spécifiques de *SpatiaLite*).

On note l'existence du projet *Microcosm* (http://wiki.openstreetmap.org/wiki/Microcosm) qui propose une implémentation en PHP de l'API OSM (dans sa dernière version 0.6) basée sur le stockage des données dans une base de données *SQLite*/*SpatiaLite*.

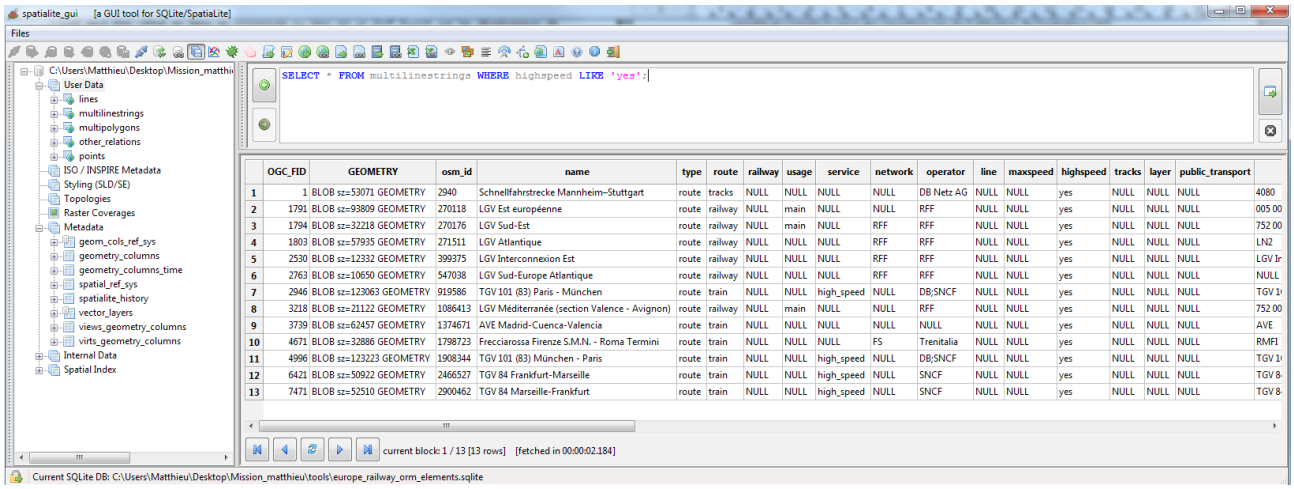


Illustration 1: Capture d'écran de Spatialite_gui

5 https://www.gaia-gis.it/fossil/*spatialite*-tools/index
6 https://www.gaia-gis.it/fossil/*spatialite*-tools/wiki?name=OSM+tools
7 Les dernières versions sont disponibles à l'adresse http://www.gaia-gis.it/gaia-sins/ (Consulté le 02/06/2015)

L'utilisation pour le traitement des données issues d'OpenSteetMap

SpatiaLite a été une des solutions retenues pour contenir les informations extraites d'OSM. Sa facilité d'utilisation et de mise en œuvre permet une utilisation parfois moins contraignante que l'utilisation de Shapefile. De plus ce format est supporté par de nombreux outils libres (*Qgis*, *OpenJUMP,* module python, etc.) ou non (*ArcGIS Desktop*, etc.).
Enfin, l'outil *ogr2ogr*, également utilisé dans le cadre de l'extraction de données OSM, base une partie de ses opérations sur la création de tables virtuelles au format SQLite/*SpatiaLite* et il permet une vitesse d'écriture ainsi qu'un poids de fichiers raisonnables par rapport à d'autres format (comme le shapefile). L'outil *ogr2ogr* supporte ainsi pleinement *SpatiaLite* et permet d'extraire des données provenant notamment (mais pas seulement) d'OSM vers une BD *SpatiaLite* conforme aux différents standards (création d'index spatiaux, supports des métadonnées, etc.).

Parmi les extensions de *SpatiaLite* telles que *OSM-tools*, seul l'outil spatialite_osm_net a été testé , les autres ne correspondants pas aux besoins voulus (en effet les outils *spatialite_osm_raw* ou *spatialite_osm_map* n'incluent pas de filtre et visent à importer l'ensemble des données, de manière la plus conforme au modèle de données OSM pour *_raw*, et au contraire d'une manière conventionnelle en SIG avec *_map* en rassemblant les points en 3 tables selon leur géométrie : point, ligne ou polygone). L'outil spatialite_osm_net réalise l'extraction et la préparation des données OSM afin d'obtenir un graphe topologique permettant des calculs d'itinéraires selon 2 modes, *rail* et *road*. Les données obtenues peuvent être utilisées avec le module *VirtualNetwork*, un module de *routing* utilisant les algorithmes *A** ou *Dijkstra* et proposant différentes options (prise en compte des sens uniques, calcul de distance ou de distance-temps).

On note également l'existence d'un plugin Qgis, *OSMDownloader*, qui, en plus de permettre le télécharger du fichier OSM d'une zone choisie, permet le traitement des fichiers OSM au format xml pour les importer dans une base de données *SQLite/SpatiaLite*.

Différentes manipulations ont ainsi pu être effectuées directement via SQLite/*SpatiaLite* (notamment dans le GUI de *SpatiaLite*, cf. illustration 1 mais également via Qgis qui dispose de la capacité de lire et d'écrire ce format), c'est par exemple le cas de la récupération de coordonnées pour l'ensemble des communes françaises présentes dans le base "flux mobilité 2010" de l'INSEE (cf. détails des manipulations en fiche annexe) et de l'accrochage des points correspondants à ces communes sur le réseau routier (*idem*). C'est également à partir d'une base de données de ce type qu'ont pu être effectuées des statistiques sur les données concernant le réseau ferrée en France dans OSM.
La possibilité de communiquer facilement avec la base de données (notamment en utilisant le langage python comme interface) a également permis de l'alimenter en direct lors du questionnement d'API spécialisées comme *Nominatim*[8] ou *Photon*[9].

Optimiser ses performances dans le cadre de nos traitements

Différentes modifications peuvent être effectuée via l'instruction spéciale PRAGMA qui permet d'accéder aux données internes de *SQLite* et de modifier différents différents de ces paramètres.
L'instruction sous sa forme la plus simple est de la forme `PRAGMA pragma_name;` pour connaître la valeur du paramètre et `PRAGMA pragma_name=xxxx;` pour le modifier.

Les principaux paramètres à relever sont :

- `PRAGMA case_sensitive_like = ON/OFF;` (ce paramètre indique à *SQLite* s'il est nécessaire de

8 Voir wiki.openstreetmap.org/wiki/FR:Nominatim
9 Voir https://github.com/komoot/photon et photon.komoot.de

prendre en compte la casse des caractères lors de l'utilisation de l'opérateur LIKE)

- PRAGMA cache_size = xxxxx; (*défaut : 2000*. Ce nombre peut être fortement augmenté puisqu'il correspond au nombre de page pouvant être stocké dans la mémoire vive : 2000 pages de 1024bytes représentent 2MB de mémoire RAM. Définir la valeur de ce paramètre a 2048000 allouera ainsi 2GB de mémoire vive au fonctionnement de la base de données)
- PRAGMA page_size = xxxxx; (*défaut : 1024*. Ce nombre peut-être augmenté, jusqu'à 32768, dans la limite où le cache_size est cohérent avec la quantité de mémoire RAM disponible)
  Le changement de la valeur page_size doit être suivit de la commande VACUUM; pour être prise en compte. On note également que, si elle n'est pas reprécisée, la valeur retrouvera sa valeur d'origine après une autre saisie de la commande VACUUM;
- PRAGMA synchronous = OFF; (Cette option permet de désactiver la synchronisation du journal *SQLite*. Cela permet de réduire considérablement le temps de la requête mais n'est pas sans risque pour l'intégrité de la BD si des problèmes étaient rencontrés pendant la transaction)

D'autres pratiques sont à adopter si la question de la performance est primordiale (notamment lors de l'écriture dans la base de donnée). En effet, lors d'un nombre important d'INSERT ou d'UPDATE, les performances seront bien meilleures si l'ensemble des modifications est écrit d'un bloc inscrit dans une transaction (délimitée par BEGIN TRANSACTION; et COMMIT TRANSACTION;) que si chaque nouvelle modification/entrée est soumise à l'écriture.

Bien que l'administration de ce type de base de données soit très légère, il peut être utile de réaliser une opération de nettoyage de l'espace libérable dans la base de données. Cette opération est réalisable via la commande : VACUUM;

Enfin l'utilisation des index, qu'il soient spatiaux ou non, permettra un gain de performances non négligeable. Toutefois il est nécessaire de préciser à *SpatiaLite* l'utilisation de l'index spatial lors de la requête. On note également que la création d'un index sur un INTEGER PRIMARY KEY est contre-productif et que cette colonne est déjà indexée par *SQLite* lorsqu'elle existe. Enfin la question de l'ordre des tables lors de l'écriture de certaines requêtes n'est pas à négliger : le fonctionnement interne de *SQLite* implique un passage en revu de l'ensemble des éléments de la première table à droite de la clause FROM.

Source et lien utiles

[1]www.sqlite.org
[2]www.gaia-gis.it/gaia-sins/*spatialite*-sql-4.2.0.html
[3]http://sqlitebrowser.org

L'API Overpass

Statut : Logiciel libre (Licence Affero GPL v3)
Site web de référence : http://overpass-api.de
Développé/maintenu par : Principalement par Roland Olbricht (mail : roland.olbricht@gmx.de)
Version actuelle : 0.7.50

Présentation de l'outil

L'Overpass API[4] est un système de requête, disposant de son propre langage (le langage *Overpass QL*) ou supportant le langage xml, permettant d'accéder, en lecture seule, à l'intégralité des données contenue dans la base de données OSM, la réponse à la requête étant renvoyée dans un des trois formats spécifiés (xml, json ou csv).
Les données accessibles via cet outils bénéficient de mises-à-jour quotidiennes et l'outil supporte la génération de *changefile* depuis sa version 0.7.50.
On note que plusieurs services web[10] sont basés sur le questionnement en temps réel de l'Overpass API et le traitement/affichage des données dans un navigateur web.

Utilisation basique

Les langages supportés ainsi que l'utilisation de l'outil sont décrits dans plusieurs fichiers de documentation (notamment [5] et [6]). Les requêtes peuvent être testées sur le site *overpass turbo*[7] en effet ce dernier permet de visualiser directement (sous réserve que la quantité de données soit raisonnable) les résultats sur un fond OpenStreetMap.

L'outil Overpass Query Form[8] permet de mettre en forme et de convertir les requêtes entre différents langages (XML, pretty QL, compact QL, etc.), permettant par exemple d'obtenir un lien cliquable à partir d'une requête saisie dans un formulaire.

Les sorties s'effectuant aux formats json, xml (.osm), ou csv, l'utilisation de l'API Overpass n'exclue pas nécessairement de passer ensuite par un outil de conversion de données (tel qu'*ogr2ogr* ou osmium) afin de pouvoir manipuler les données dans un format qui nous est habituel (shapefile, base de données spatiale, etc.).

Parmi les fonctions clés de cet outil, on relève la présence des fonctions récursives (qui vont permettre de récupérer les membres d'un élément (*recurse down*), ainsi que les relations auxquelles il appartient (*recurse down relations*), les éléments qui le comprennent comme un membre (*recurse up*) ou les membres de la relations décrite par l'élément (*recurse up relations*)), la fonction *foreach* (qui permet d'appliquer une requête à chaque éléments du jeu de données) ainsi que la fonction *pivot* (qui va permettre de choisir l'élément du type choisi qui définit le contour d'une aire). De plus l'action `out;` effectuée à l'issue de la requête peut être modifiée afin de choisir le niveau de détail des éléments exportés (`out geom;` permettra d'obtenir la géométrie complète des éléments et de leurs membres rendant possible une utilisation dans un SIG / `out ids;` ne retournera que les identifiants OSM / `out meta;` retournera l'ensemble des informations connues à propos des éléments, y compris les informations concernant sa dernière modification).

Ainsi une requête, rédigée en *Overpass QL*, visant à récupérer la ou les gare situées dans un rayon de 3km autour du centre de chaque ville taguée *city* pourrait prendre la forme :

10 http://wiki.openstreetmap.org/wiki/Overpass_API/Applications

```
[out:json][timeout:1500];
node({{bbox}})[place=city];
foreach->.a
 (
  node(around.a:3000)[railway=station][station!~'subway|tram'];
    out;
 );
out;
```

Différentes modifications pourraient être mise en place pour affiner cette recherche, telles que l'ajout de `[operator~'SNCF|SBB|DB|NMBS|SNCB|CFL|RFI']` à la ligne 5 (pour préciser que seules les stations exploitées par l'exploitant national, ici Français, Suisse, Allemand, Belge, Luxembourgeois et Italien, sont recherchées),
et/ou
`out 1;` à la ligne 6 (pour ne renvoyer qu'une seule gare pour chaque lieu *place=city* trouvé, attention toutefois au fait qu'il ne s'agira pas forcément de la plus proche ou de la plus importante).

Les raisons de l'utiliser / Les utilisations qui en ont été faites

L'outil overpass turbo (illustration 2) est un outil précieux lorsqu'il s'agit d'effectuer des tests "à la volée" sur des zones restreintes ou des éléments limités, que ce soit avant une requête plus importante à l'API ou afin de vérifier des informations sur des zones restreintes.

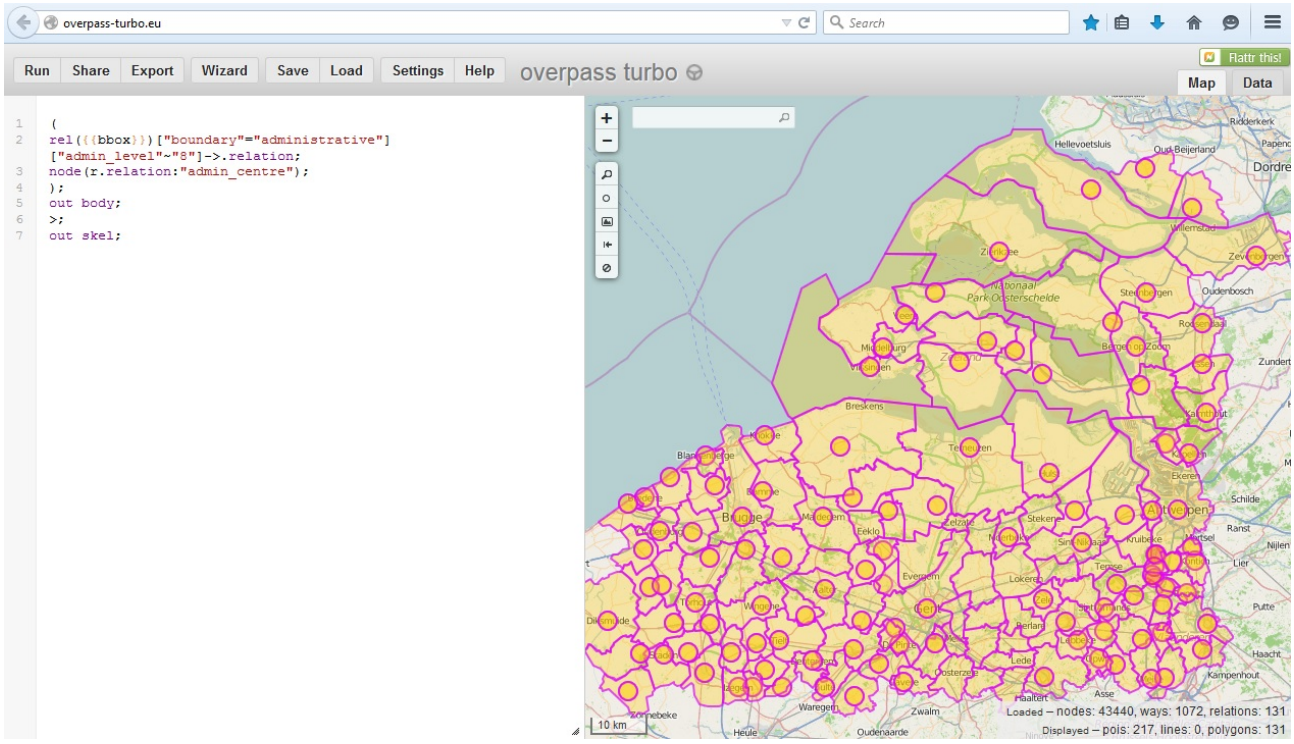


Illustration 2: Capture d'écran du site web overpass-turbo.eu

Différentes requêtes, s'intéressant par exemple à des éléments au sein d'une large aire spatiale (l'Europe par exemple) mais à un nombre limité d'éléments (les capitales des pays par exemple) ou à des objets à la géométrie simple (des points par exemple), sont à la faveur de l'API Overpass et de la réactivité de ses serveurs. Il est en effet beaucoup plus simple dans ces cas précis de construire une requête et de télécharger/traiter le fichier contenant les résultats que de télécharger le jeu de données complet pour

l'ensemble de la zone (15GB pour l'Europe[11] par exemple) puis de le traiter.

Au contraire, des requêtes complexes, comprenant beaucoup d'éléments en sortie (et des géométries complexes et détaillées) généreront des fichiers très volumineux en sortie de l'API et/ou ne seront pas traiter en raison d'un *timeout* du serveur. Des requêtes de ce type (extraction de l'ensemble du réseau routier de l'Europe par exemple) vont à la faveur d'outil tels que *osmfilter*, *osmosis* ou *ogr2gr* par exemple.

L'Overpass API se veut également être un moyen de créer un lien permanent vers un objet OSM. En effet les ID des éléments peuvent être amenés à changer, risquant ainsi de rompre un lien tel que centre de la commune de Mende (http://www.openstreetmap.org/node/26691841). L'Overpass API propose ainsi, grâce à la combinaison habile de tags de la part de l'utilisateur, de proposer un lien direct vers l'élément s'il a été trouvé. Ainsi le lien permanent à donner pour désigner le centre de la commune de Mende pourrait être : http://overpass-api.de/api/interpreter?data=[out:custom];node[place=town]["name:fr"="Mende"];out;

Sources et liens utiles

[4] http://overpass.de
[5] http://wiki.openstreetmap.org/wiki/Overpass_API
[6] http://wiki.openstreetmap.org/wiki/Overpass_API/Language_Guide)
[7] http://overpass-turbo.eu
[8]http://overpass-api.de/query_form.html

11 http://download.geofabrik.de/europe.html. Consulté le 18/05/2015.

ogr2ogr

Statut : Logiciel libre (Licence type X/MIT par l'*Open Source Geospatial Foundation*)
Site web de référence : http://www.gdal.org/*ogr2ogr*.html
Développé/maintenu par : Communauté GDAL/OGR
Version actuelle : 1.11
(la version 2.0 est disponible en bêta-test et apporte de nombreuses fonctionnalités)

Présentation

(Une partie des fonctionnalités décrites s'applique également à l'outil *ogrinfo* qui permet, non pas de convertir des éléments, mais d'obtenir des informations détaillées sur des éléments, tout en utilisant un système de requête semblable à *ogr2ogr*)

L'outil s'utilise uniquement en ligne de commande et permet de convertir des données entre différents formats (83 formats sont supportées, dont une partie seulement en lecture[12]), tout en opérant différentes opérations lors de la conversion, telles que des sélections (spatiales ou attributaires) ou des reprojections. Ainsi une source de données externes pourra aussi bien être intégrée dans une base de données existantes (*PostGIS, SQLite, Oracle Spatial*, etc.) qu'exportée directement en Shapefile, en GeoJson ou sous forme d'un document Geospatial PDF.

L'outil ogr2ogr se base sur la bibliothèque logicielle GDAL/OGR, écrite en C/C++. Cette bibliothèque est utilisée par de nombreux logiciels (Qgis, GRASS GIS, etc.), est accessible par différents langages de programmation (C++, Java, Perl, Python, R, Ruby, etc.) et supporte l'utilisation et le traitement de fichiers volumineux.

Cet outil utilise un système de requête autorisant le langage SQL (la principale capacité implémentée est SELECT) [9], quel que soit le format qui soit lu et écrit dans la mesure où il est supporté par GDAL. Cette fonctionnalité est enrichie (depuis la version 1.10) par le support du "dialecte" *SQLite* [10], permettant ainsi de bénéficier de requêtes étendue (UPDATE, INSERT, DELETE) ainsi que des fonctions spatiales de *SpatiaLite* (via le champ GEOMETRY qui est rendu actif et manipulable).

L'utilisation dans le cadre du traitement des données issues d'OpenStreetMap

L'outil *ogr2ogr* va permettre de lire les données OpenStreetMap [11] (aux formats .osm ou .osm.pbf), d'y effectuer des sélections et des transformations (qu'elles soient spatiales ou attributaires) avant de les sauvegarder dans le format voulu. Cette fonction est réalisée grâce à l'analyse des données OSM sur la base d'un dictionnaire d'attributs préalablement défini (il s'agit du fichier *osmconf.ini* dont une version standard est créée à l'installation des outils gdal/ogr) puis par la création de table virtuelles [12] au format *SQLite* sur lesquelles sont effectuées les éventuelles requêtes saisies, avant d'écrire les donnés sélectionnée au format souhaité.

Le traitement du format OSM par *ogr2ogr* s'effectue par une décomposition des éléments en 5 tables virtuelles :

- *points* : les éléments de type "*node*"
- *lines* : les éléments de type "*way*"
- *multilinestrings* : les éléments de type "*relation*" composés de plusieurs lignes (et correspondant à

12 La liste complète et les détails sont disponibles sur : http://www.gdal.org/ogr_formats.html (Consulté le 02/06/2015)

type= multilinestring ou type=route dans OSM)

- *multipolygons* : les éléments de type "*relation*" formant une surface/un polygone (et correspondant à type=multipolygon ou type=boundary dans OSM)
- *other_relations* : les éléments de type "relation" ne correspondant pas aux catégories précédentes (ces éléments comprennent généralement des éléments hétéroclites : points et lignes par exemple, et correspondent alors au format *GeometryCollection* de l'OGC, dans OSM ce sont généralement les différents éléments appartenant à un site, tel qu'un aéroport, qui y sont regroupés)

Bien que les champs attributaires des éléments créés correspondent à ceux définis dans le fichier *osmconf.ini*, puis éventuellement à ceux sélectionnés lors de la saisie de la commande, *ogr2ogr* offre la possibilité de récupérer les tags ne figurant pas dans le fichier de configuration pour les stocker dans le champ attributaire *other_tags* (ou l'ensemble des tags dans le champ *all_tags*) ; ils sont ainsi stockées/trier d'une manière comparable à l'utilisation de *hstore* dans *PostgreSQL*.

Bien que cet outil ne soit pas l'outil de référence en terme d'extraction de données OSM notamment car il ne reproduit aucun des schémas classiques de base de données utilisés par OSM (voir [13] pour consulter la liste des schémas existants), ne conservant ainsi pas l'ensemble des relations et ne lui permettant pas de supporter la gestion des mise à jour à partir des *changefile* OSM, il s'avère être utile aussi bien dans l'extraction thématique de données que dans la construction d'une base de données constituée de différents types d'éléments. Ses différentes fonctionnalités (et notamment son système de requête) le rendent adapté à une utilisation dans un cadre d'analyse/de fouille de données ainsi qu'en vue d'obtenir des données thématiques à cartographier.

- Exemple d'utilisation visant à importer dans une base de données SQLite les informations relatives au transport ferroviaire (sur la base d'un fichier filtré, cf. exemple fournit dans la fiche osmfilter) :

```
> ogr2ogr -f SQLITE europe_railway_orm_elements.sqlite -dsco
SPATIALITE=YES -lco ENCODING=UTF-8 --config
OGR_INTERLEAVED_READING YES --config OSM_CONFIG_FILE ..\..\
Encours\osmconf.ini-railway_only -gt 65536 --config
OGR_SQLITE_CACHE 2048 --config OSM_MAX_TMPFILE_SIZE 2048 ..\
donnees-osm\europe_filt_railway_orm.osm.pbf
```

- Exemple d'utilisation visant à exporter en shapefile les informations concernant la localisation des mairies :

```
> ogr2ogr -f "ESRI Shapefile" townhall_output_folder -dialect
SQLite -sql "SELECT * FROM points, multipolygons WHERE
points.amenity LIKE 'townhall' OR multipolygons.amenity LIKE
'townhall'" --config OGR_INTERLEAVED_READING YES -lco
ENCODING=UTF-8 --config OSM_CONFIG_FILE C:\OSGeo4W64\share\
gdal\osmconf.ini --config OSM_MAX_TMPFILE_SIZE 2048 france-
latest.osm.pbf
```

- Exemple d'utilisation d'ogrinfo basée sur une requête construite en SQL (visant ici à connaître le nombre de points tagués 'city' , quel que soit le nombre d'habitants, et ceux tagués 'town' seulement si le champ population est supérieur à 100000) :

```
> ogrinfo -sql "SELECT * FROM points WHERE (cast(population as
INTEGER)> 100000 AND place LIKE 'town' ) OR place LIKE 'city'"
--config OSM_CONFIG_FILE C:\OSGeo4W64\share\gdal\osmconf.ini
france-latest.osm.pbf
```

Optimisation de l'utilisation des drivers SpatiaLite et OSM

Différentes options existent et permettent d'accélérer le traitement par *ogr2ogr* (le nom des options est ici donné pour une version de GDAL/OGR < 2.0) :

- -dsco *SPATIALITE*=YES : cette option est la première à considérer lors de l'écriture de fichier au format SQLite/*SpatiaLite*, elle va permettre la création de tables "valides" selon *SpatiaLite* (stockage de la géométrie au format SQL, création d'index spatiaux et des tables de métadonnées (*geometry_columns* et *spatial_ref_sys*) , définition de la projection utilisée).
- --config OSM_MAX_TMPFILE_SIZE xxxx (ou xxxx est un chiffre en Mo) : cette option permet de spécifier la quantité de mémoire RAM à allouer au traitement du fichier OSM (par défaut cette valeur est de 100), une fois cette capacité atteinte c'est de l'espace disque (via des fichiers temporaires) qui est utilisé.
- --config OGR_SQLITE_CACHE xxxx (ou xxxx est un chiffre en Mo) : cette option permet de spécifier la quantité de mémoire RAM à allouer à l'écriture de la base de données SQLite.
- --config OGR_INTERLEAVED_READING YES : cette option permet de spécifier
- --config OGR_SQLITE_SYNCHRONOUS OFF : cette option, dont l'usage peut s'avérer risqué en cas de problème de la machine lors de l'exécution d'*ogr2ogr*, désactive toutes les opérations de synchronisation, permettant un cas non négligeable de performances.
- --config OSM_CONFIG_FILE /path/to/osmconf.ini : cette option permet de spécifier le chemin du fichier *osmconf.ini* à utiliser.
- -gt xxxx (où xxxx est un chiffre en bytes) : cet argument permet de spécifier explicitement le nombre de ligne à écrire au cours de chaque transaction (une valeur de 65536 permet d'assurer des performances optimales lors de la création de tables importantes).
- -skipfailures : cette option peut s'avérer nécessaire lorsque la géométrie à insérer dans la table SQLite ne respecte pas certaines contraintes (cette option risque toutefois de causer une importante perte de performances).
- -explodecollections : cette option permet de décomposer les collections de géométries vers les éléments qui les composent (*points*, *lines*, *multilinestrings* et *multipolygons*), elle peut être appliquée lors du traitement d'un fichier OSM (les éléments restent dans la tables *other_relations* mais y sont décomposés) ou à posteriori sur cette table *other_relations* pour ne sélectionner qu'un seul type de géométrie tel que des points.

Solutions alternatives

Différentes autres solutions existent pour exporter les données OSM vers une base de données. Le choix d'une solution par rapport à une autre se fera en fonction de l'utilisation prévue (analyse de données, représentation cartographique, installation d'une API basée sur OSM, etc.) ainsi que de la plate-forme choisie pour accueillir les données (base de données, fichiers textes, shapefile, etc.) et l'on se référera notamment à la documentation officielle d'OSM [13].
Les solutions les plus utilisées sont notamment *osm2pgsql*, *imposm* et *osmosis* (ces 3 derniers permettent notamment l'importation des données OSM dans une base de données *PostGreSQL/PostGIS*, selon des schéma utilisé par les principaux services tels que MapNik, Nominatim ou l'API OSM) ainsi que *MongOSM* et *goosm* (permettant tous 2 l'importation des données vers une base de données *MungoDB*).

On note également qu'un plugin Qgis s'appuie sur *ogr2ogr* : le plugin quickOSM [14] qui offre une interface graphique pour envoyer des requêtes à l'overpass API ou pour utiliser un fichier OSM local et le visualiser dans Qgis et/ou l'exporter vers les format gérés par Gdal/ogr.

Sources et liens utiles

[9] http://www.gdal.org/ogr_sql.html

[10] http://www.gdal.org/ogr_sql_sqlite.html
[11] http://www.gdal.org/drv_osm.html
[12] http://www.gdal.org/drv_vrt.html
[13] http://wiki.openstreetmap.org/wiki/Databases_and_data_access_APIs
[14] https://plugins.qgis.org/plugins/QuickOSM/

osmfilter / osmconvert / osmupdate

Statut : Logiciel libre (Licence Affero GPL v3)
Site web de référence : https://gitorious.org/osm-c-tools et les 3 pages dédiées du wiki OpenStreetMap
Développé/maintenu par : Markus Weber (mail : markus.weber@gmx.com)
Version actuelle : osmfilter : 1.3 / osmconvert : 0.8.2 / osmupdate : 0.4.1

Présentations

Ces trois outils (regroupés sous le package *osmtools* sur debian/ubuntu et parfois désignés sous le nom *osm-c-tools*) sont codés en C et s'utilisent uniquement en ligne de commande. Ils se distinguent par leur rapidité et vont permettre d'effectuer des transformations sur des fichiers au format osm.

Osmconvert permet d'effectuer des conversions entre la plupart des formats (.osm, .osc, .osc.gz, .osh, .o5m, .o5c, .osm.pbf). L'outil permet également d'effectuer une extraction d'une zone (définie par son emprise sous forme d'un polygone au format .poly) à partir d'un fichier à plus large emprise. De plus l'outil permet d'appliquer les fichiers de mise à jour d'OSM (*changefile* au format .osc ou .o5c) sur un fichier standard (.osm.pbf par exemple) ou d'extraire le fichier des différences issu de la comparaison de deux fichiers OSM. Enfin, différentes options sont proposées pour produire des statistiques sur le fichier en entrée (récupération du *timestamp*, statistiques sur le nombre d'éléments par type). On note que cet outils permet également d'écrire vers le formats .csv.

Osmfilter permet de filtrer les éléments présents dans un fichier OSM (aux formats .osm ou .o5m d'où l'utilisation parfois nécessaire de l'outil précédent pour effectuer la conversion), selon différents critères : leurs types, leurs tags (et les valeurs qu'ils prennent), leurs métadonnées ou leur rôle au sein d'une relation. Ce filtrage va s'effectuer par différentes requêtes (visant soit à ne conserver que certains éléments, soit à n'éliminer que certains éléments) pouvant utiliser des opérateurs complexes (opérateurs booléens, *wildcards*, utilisation des tags spéciaux pour filtrer sur la base des métadonnées). Cet outil permet également de n'obtenir que des statistiques sur les tags des éléments présents dans le fichier OSM fournit en entrée (statistiques sur les clés et valeurs rencontrées, liste des valeurs prises par une donnée et trie par le nombre d'occurrences).

Osmupdate permet de télécharger et d'appliquer des mises à jours à partir du serveur principal OSM ainsi que de générer des *changefile* avec différentes options concernant la fréquence de ces opérations (création d'un nouveau *changefile* pour chaque heure à partir d'un fichier OSM existant, mise à jour d'un fichier OSM en vue de garder une base de données PostgreSQL à jour en quasi-permanence, etc.)

Ces trois outils permettent des performances intéressantes par rapport à certains de leurs concurrents et il peut être approprié de les utiliser, par exemple, pour effectuer un premier comptage des éléments d'intérêts sur un large fichier ou pour effectuer un premier filtrage avant d'utiliser un outils d'extraction des données OSM vers une base de données (il est en effet plus rapide d'effectuer un premier filtrage avec *osmfilter* puis d'extraire les données avec *ogr2ogr* que de laisser *ogr2ogr* s'occuper du filtrage et de l'extraction). De plus, les relations relatives aux éléments filtrés sont conservées et *osmfilter* est donc un bon candidat pour un filtrage préalable à l'utilisation de différents outils d'extraction.
La combinaison du trio *osmfilter/osmconvert/osmupdate* permet de réaliser des opérations complexes (par exemple mise à jour régulière d'une base OSM puis filtrage des données d'intérêt) avec des commandes simples. On peut noter que ce trio d'outils est utilisé (en combinaison avec *osm2pgsql*) pour extraire les éléments nécessaires à la visualisation fournit par le projet *OpenRailwayMap* [18] [19].

L'utilisation dans le cadre du traitement des données issues d'OpenStreetMap

Quatres exemples d'utilisation d'osmconvert:

- Conversion:
```
> osmconvert file.osm.pbf -o=output.o5m
```
- Extraction d'une zone :
```
> osmconvert large_file.osm.pbf -B=boundingarea.poly -
o=output.osm.pbf
```
- Mise à jour d'un fichier .osm existant :
```
> osmconvert europe-old.osm changefile.osc -o=europe-new.osm
```
- Récupération des différences entre deux versions d'un fichier OSM :
```
> osmconvert old.osm new.osm --diff -o=changefile.osc
```
- Lecture du timestamp d'un fichier OSM :
```
> osmconvert europe-latest.osm.pbf --out-timestamp
```

Deux exemples d'utilisation d'osmfilter :

- Comptage du nombre d'éléments portant la clé *place* en France et des valeurs qui lui sont attribuée :
```
> osmfilter france.o5m --out-count=place
```

- Filtrage du fichier OSM d'une zone donnée (ici l'Europe, par exemple après l'application), pour ne conserver que les éléments relatifs au transports ferroviaire et réduire le temps nécessaire à l'outil réalisant l'extraction ou à l'outil servant à visualiser les données directement depuis le fichier OSM :
```
> osmfilter europe.o5m --keep="railway= route=tracks
route=railway route=train route=light_rail route=tram
route=subway route_master=train route_master=light_rail
route_master=tram route_master=subway shop=ticket
vending=public_transport_tickets" -
o=europe_railway_elements.o5m
```

Trois exemples d'utilisation d'osmupdate :

- Mise à jour d'un fichier OSM tout en appliquant une limite de l'emprise géographique :
```
> osmupdate planet_old.o5m planet_new.o5m -B=study_area.poly
```
- Mise à jour d'un fichier OSM :
```
> osmupdate europe_old.o5m europe_new.o5m
```
- Générer un *changefile* entre la version ancienne présente localement et la version à jour :
```
> osmupdate europe.o5m changefile.o5c
```

Solutions alternatives

Les traitements proposés par ces trois outils peuvent être réalisés avec d'autres outils tels que O*smosis* [20] (pour le découpage d'un emprise particulière, le filtrage de clés/valeurs et la mise à jour des fichiers OSM), *Osmium* [21] (conversion, application de *changefile*, extraction d'informations à partir d'un fichier OSM historique), *ogrinfo* (pour obtenir des informations plus approfondies sur les éléments présents dans un fichier OSM) ou *ogr2ogr*. Ces solutions offrent des options parfois différentes et sont également à prendre en considération lors de la manipulation de données OSM.

Sources et liens utiles

[15] http://wiki.openstreetmap.org/wiki/Osmfilter
[16] http://wiki.openstreetmap.org/wiki/Osmconvert
[17] http://wiki.openstreetmap.org/wiki/Osmupdate
[18] http://www.openrailwaymap.org/
[19] https://github.com/rurseekatze/OpenRailwayMap/
[20] http://wiki.openstreetmap.org/wiki/Osmosis
[21] http://osmcode.org/osmium/

L'installation et l'utilisation en local de l'instance d'une API spécialisée

L'installation puis le questionnement en local d'une API spécialisée fait partie des bonnes pratiques à adopter lorsque de nombreuses requêtes sont à effectuer, afin notamment de pas surcharger le serveur receveur et de ne pas occasionner de trafic réseau superflu.

Plusieurs API liées à OpenStreetMap peuvent être déployées localement. Ces API peuvent avoir des finalités variées :

- géolocalisation (Nominatim, Photon, Pelias, etc.)
- routage/routing (OSRM, GraphHopper, Valhalla, etc.)
- obtention d'information sur les éléments OpenStreetMap (Overpass API, Taginfo API, etc.)
- création de cartes/fonds basés sur OSM (OpenLayers, etc.)

Exemple n°1 : L'engin de routage OSRM

Open Street Routing Machine (OSRM)[22] est un projet libre (sous licence type BSD) combinant des algorithmes de routage avancés avec les données du réseau routier OpenStreetMap. Une implémentation de ce projet est proposée depuis sur la page officielle du projet OpenStreetMap pour réaliser des calculs d'itinéraires en voiture (accessible sur la page www.openstreetmap.org dans l'encart de recherche, en cliquant sur le bouton bleu représentant deux flèches). Le projet est principalement maintenu par Dennis Luxen et Christian Vetter.

Le projet, développé en C++, permet le calcul du plus rapide chemin entre deux lieux avec la récupération des informations de parcours, le calcul de matrice de temps de parcours entre une liste de lieux, le calcul de parcours avec plusieurs points intermédiaires, l'accrochage au réseau de points issues d'une trace GPS, etc.
La principale spécificité d'OSRM se situe dans l'utilisation de la méthode de la contraction des hiérarchies (ce concept est notamment tiré d'une thèse publié en 2008 [23]), permettant un pré-traitement du graphe du réseau routier, rendant ainsi possible des recherches plus rapides. Une fois ce réseau préparé l'algorithme utilisé n'est pas l'algorithme A* (*A-star*) mais une variante de l'algorithme de *Dijkstra*.

Une fois les données extraites et préparée, l'instance locale du serveur peut être questionnée à l'adresse :

http://localhost:5000/{service}?{parametre}={valeur}

L'API implémente partiellement le standard HTTP 1.1 (à l'exception notable du *pipelining*) et les réponses fourni sont au format json [24].

Exemple n°2 :Le service de géocodage Photon

Photon[25] est un système de géocodage conçu pour les données OpenStreetMap et basé sur le moteur de recherche Elasticsearch[26].
Photon est un outil multi-plateforme placé sous licence Apache v2.0. Il est développé en Java et ne nécessite pas d'installation à proprement parlé. L'outil se base sur la construction d'un index (basé sur le schéma de Nominatim) et propose une API plus flexible que Nominatim (prise en compte d'un biais de localisation, prise en compte des erreurs de typographie, recherche multi-langues, etc.).
Contrairement à d'autres API du même type, Photon propose de télécharger directement l'index utilisé pour réaliser les recherches (environ 31 GB compressé), représentant ainsi un gain de temps non négligeable (la préparation de Nominatim, à partir du fichier *planet.osm*, peut durer plusieurs jours), au détriment d'un certaine flexibilité (contrairement au déploiement d'une base de données d'après le schéma

utilisé par Nominatim, cet index ne pourra en effet pas bénéficier de mise à jour et ne rend pas possible le prise en compte d'autres langues que les 4 prévues).

Une fois l'index téléchargé et l'instance de Photon lancée, le service peut être questionné par des requêtes de la forme :

- Requête de base :
  http://localhost:2322/api?q=strasbourg

- Introduction d'un biais de localisation :
  http://localhost:2322/api?q=berlin&lon=10&lat=52

- Spécification d'une limitation du nombre de résultats et recherche portant sur des tags spécifiques (ici les gares) :
  http://localhost:2322/api?q=Mende,France&osm_tag=railway:station&limit=1

- Spécification de la langue dans laquelle sont retournées les résultats :
  http://localhost:2322/api?q=strasbourg,france&lang=de
  (ici *Strasbourg, France* retournera les noms en allemand : *Straßburg, Elsass, Frankreich*)

- Reverse geocoding :
  http://photon.komoot.de/reverse?lon=1.455769&lat=45.190758

Exemple n°3 :L'obtention d'informations et de statistiques sur les tags utilisés avec l'API taginfo

L'API taginfo va permettre de trouver, d'agréger et de mettre à disposition des informations à propos des tags utilisés. L'instance du projet principal (appliqué à l'ensemble de la base de données) propose de visualiser les résultats de manière interactive dans un navigateur web et est consultable à l'adresse http://taginfo.openstreetmap.org/. Des déclinaisons locales limitant l'emprise de l'outil existent également pour plusieurs pays [27].
L'API est également interrogeable directement : les fonctions proposées sont riches (voir la documentation complète [28]), les requêtes permettent de filtrer directement selon le type d'élément (*node*, *way*, *relation*) recherchés et les résultats sont fournis au format JSON.
Cet outil est utile par exemple afin de savoir quels sont les tags à choisir (et sur quels type d'élément ils s'appliquent) avant de réaliser une extraction thématique, mais il permet aussi de suivre l'évolution de tags d'intérêt (par exemple vérifier l'utilisation de tags tombés en désuétude et censés disparaître), d'afficher des cartes de répartition des éléments et fait le lien vers les pages du wiki OSM décrivant la clé (ou le couple clé=valeur) souhaité.
Afin de savoir quels tags extraire après avoir identifié les aéroports comme les objets tagué *aeroway=aerodrome* il est possible d'interroger l'API pour connaître les combinaisons utilisées :
http://taginfo.openstreetmap.org/api/4/tag/combinations?
key=aeroway&value=aerodrome&sortname=together_count&sortorder=desc

Il est possible de télécharger les données utilisées par l'API [29] afin de les mobiliser localement, telles quelles ou avec les mêmes outils que ceux utilisés sur le site web du projet [30].

Différentes projets reposent sur l'utilisation des données de tag info, tels que *Tag Finder* [31][32](un moteur de recherche de tags) ou *OSMonto* [33].

Source et liens utiles :

[22]http://project-osrm.org/
[23] http://algo2.iti.kit.edu/download/diploma_thesis_geisberger.pdf
[24]https://github.com/Project-OSRM/osrm-backend/wiki/Server-api
[25] https://github.com/komoot/photon
[26] http://fr.wikipedia.org/wiki/Elasticsearch
[27] http://wiki.openstreetmap.org/wiki/Taginfo/Sites
[28] http://taginfo.openstreetmap.org/taginfo/apidoc
[29] http://taginfo.openstreetmap.org/download
[30] https://github.com/joto/taginfo
[31] http://tagfinder.herokuapp.com/
[32] https://github.com/geometalab/OSMTagFinder
[33] http://wiki.openstreetmap.org/wiki/OSMonto

L'utilisation du langage Python et de modules dédiés

Cf. document html.