Development of a Query Engine for Cross-Querying Heterogeneous Graphs: Application to Geospatial Data

3IF Internship

Nathan Lèbre

2025

1 Geographical zone

Given the complexity in terms of memory of geographical data, we need to restict our study area to a small geographical zone, which allows us to work with a local database.

The 'Doua' campus will define our study area. In OpenStreetMap, its delimited by the way 394434727.

In terms of statistics, the obtained subgraph contains

Type	Number
Ways	7401
Nodes	3523
Relations	44
Predicates	689 049

2 Dataset

2.1 OpenStreetMap

OpenStreetMap data available to export for a user-defined polygon are encoded as an XML format with a proper extension (`.osm`).

To use those data in a RDF graph, we have to convert them into a RDF serialization format (as Turtle, RDF/XML, N-Triples, JSON-LD ...).

To this end, `osm2rdf` tool can be used to convert a `.osm` into a `.pbf`, and then into a `.ttl`.
`.pbf` files are also available on Geofabrik for a specific region.

The data will therefore be accessible via SPARQL queries.

QLever endpoint can also be used in order to fetch directly the interesting triples thanks to `CONSTRUCT` SPARQL queries (the result is given in a `csv` file easily convertible into a `.nt` thanks to a Python script).

2.2 Open Meteo

Open Meteo offers meteorological historical and forecasts for a defined coordinates point, with a resolution from 1 to 2 kilometers. API provides a JSON file.

2.3 Other data sources

2.3.1 GeoNames

GeoNames lists 11 million places whose information can be freely downloaded. GeoNames is based on semantic web and therefore on a hierarchical structuring of information. GeoNames graph contains 182 million RDF triples. Consequently, it is necessary to extract the subgraphs containing the relevant information using a SPARQL `CONSTRUCT` query. However, since all the names are already present in OSM, GeoNames may not be particularly useful for us.

2.3.2 GrandLyon Data

GrandLyon Data allows us to access to various data consultation services. The list of available data sets can be found here. However, those data aren't query-able or easily link-able to OSM data.

3 OpenStreetMap

3.1 Classes

The architecture of OpenStreetMap is based on three classes at the same hierarchical level :

- Points (**nodes**)
- Segments (**ways**)
- Objects collections (**relations**)

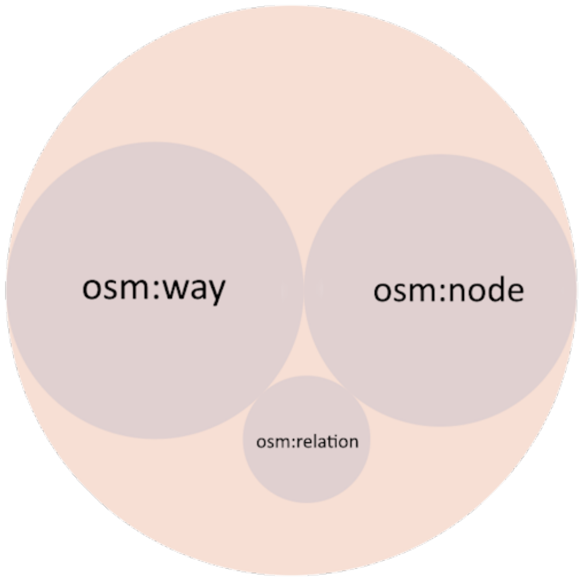


Figure 1: Classes diagram

3.1.1 Nodes

Nodes are the constituent part of OpenStreetMap.

Two nodes types can be distinguished :

- Nodes that are defining a way
- Nodes as specific point of interest

Nodes have the following properties :

Property	Type	Description
id	Integer ≥ 1	Unique node AI
version	Integer ≥ 1	Version node number, generated with each modification
visible	Boolean	Node visible or not
changeset	Integer ≥ 1	Modification group which created this version
user	String	User who created this version
uid	Integer ≥ 1	Unique user ID
timestamp	Date (ISO)	Timestamping from the last version
lat	Float [-90,90]	Latitude (WGS84)
lon	Float [-180,180]	Longitude (WGS84)

Every node also have an attribute list (tags) corresponding to key-value pairs (keys are unique).

Observation :

- Timestamp dates are in UTC, format `AAAA-MM-JJTHH:MM:SSZ`
- The URI corresponding to a predicate for the data d is `osmmeta:d`.
- Attibutes are pairs (predicate, object), where the predicate is `osmkey:k` for the key `k`.
- A specific point of interest always have the `amenity` key.

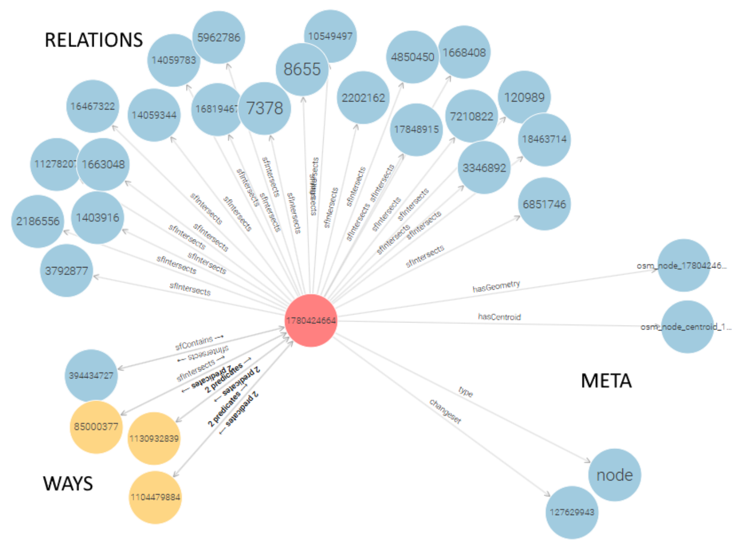


Figure 2: Example of a node subgraph

Literals are not printed

3.1.2 Ways

Ways are constituted from a list of nodes and describe roads or surfaces (closed way if the first node is equal to the last one).

Observations :

- A path can include from 2 to 2000 nodes.
- A way cannot include the same node two times (except the extremity of a closed path).

Property	Type	Description
id	Integer ≥ 1	Unique way ID
nodes	List	List of nodes composing the way

Every path also have an attribute list (tags).

Observation :

- The way ID is unique (the number can, however, be the same as a node or relation ID).
- To get the nodes composing the way : `?way ogc:sfContains ?node`.

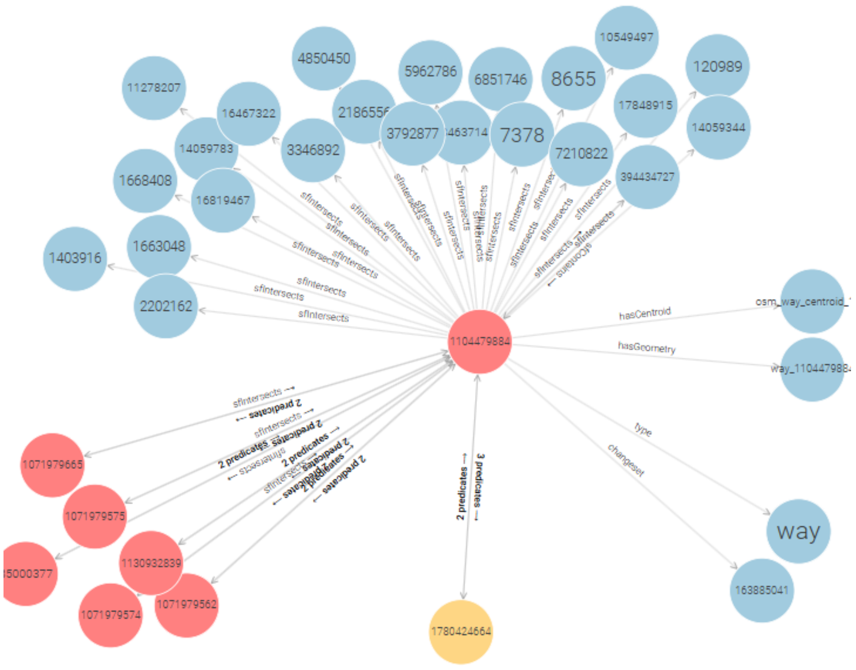


Figure 3: Example of a way subgraph

Literals are not printed

.

3.1.3 Relations

Relations is useful to gather items (nodes, ways, relations). Every relation has a `type` attribute.

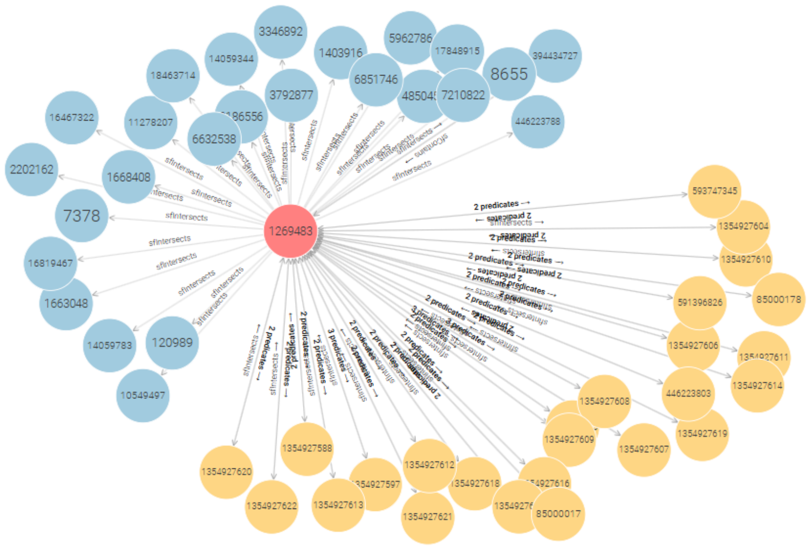


Figure 4: Example of a relation subgraph

Literals are not printed

3.2 Prefixes and URI

In order to make the queries more readable, the following prefixes can be used to write shorter URI :

Prefix	URI
osmkey	`https://www.openstreetmap.org/wiki/Key:`
osmmeta	`https://www.openstreetmap.org/meta/`
osmnode	`https://www.openstreetmap.org/node/`
osmrel	`https://www.openstreetmap.org/relation/`
osmway	`https://www.openstreetmap.org/way/`
osm	`https://www.openstreetmap.org/`
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
rdfs	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
ogc	`http://www.opengis.net/rdf#`
geo	`http://www.opengis.net/ont/geosparql#`
geof	`http://www.opengis.net/def/function/geosparql/`
osm2rdf	`https://osm2rdf.cs.uni-freiburg.de/rdf#`
osm2rdfgeom	`https://osm2rdf.cs.uni-freiburg.de/rdf/geom#`

3.3 Predicates

rdf The `rdf:type` predicate is used to give a type to an item : `rdf:type = osm:node`, `rdf:type = osm:way` or `rdf:type = osm:relation`.

ogc The predicates having an URI beginning by `ogc` are used to characterized geographical inclusion.

- `ogc:sfIntersects`: the subject element shares at least one point with the object element.
- `ogc:sfContains`: all points of the object element are inside the subject element.
- `ogc:sfTouches`: the subject element shares at least one point with the object element, but their interiors do not overlap.

Observations :
Some way-node inclusions could be also modeled in this way :
`osmway:663613002 osmway:member _:bn896700245 .`
`_:bn896700245 osmway:member_id osmnode:228934681 .`
which is equivalent to the simple `osmway:663613002 ogc:sfContains osmnode:228934681.` predicate.

geo `geo` predicates are used to describe the geometry of an object

- `geo:hasGeometry`: links an object to an element representing its full geometry (point, line, or polygon).
- `geo:hasCentroid`: links an object to an element representing its centroid (a point at the center of the geometry if it is not already a point).
- `geo:asWKT`: applied to the geometry element linked to the object (via `geo:hasGeometry` or `geo:hasCentroid`) to obtain its coordinates in WKT format.

osm2rdf `osm2rdf` and `osm2rdfgeom` predicates give additional informations about OSM objects

- `osm2rdf:facts` : provides the number of attributes or properties.
- `osm2rdf:length` : provides the length of an object (for linestrings or polygon).

- `osm2rdfgeom:convex_hull`: provides the minimal convex hull that contains the object.
- `osm2rdfgeom:envelope`: provides the minimal bounding rectangle of the geometry.
- `osm2rdfgeom:obb`: provides the oriented bounding box, often more precise than the standard rectangle.

osmkey See the dedicated section.

osmmeta `osmmeta` predicates are used to link metadata to an element (version, last modification date, user who did the most modification).Those predicates will not be useful for us.

3.4 Useful Attributes

3.4.1 General

osmkey:amenity `amenity` is the main key used to characterize useful and important establishments.
For example, possible values associated with this key are `cafe`, `restaurant`, `college`, `library`, `research_institute`, `school`, `university`, `bus_station`, `parking`, `bank`, `hospital`,

pharmacy, cinema, theatre, post_office, toilets, etc.
Observations :

- Educational buildings use the education key instead since 2025.

osmkey:name The name key is used to name objects (noname=yes for unnamed objects).

osmkey:addr Sub-keys of addr are used for points of interest and areas (if the address is the same for all contained objects). The most commonly used sub-keys are addr:housenumber, addr:street, addr:city, addr:postcode, addr:country, etc.

3.4.2 Mobility

osmkey:highway Main tag for roads:

Table 1: Common values for the highway tag

Value	**Description**
motorway	Motorway
trunk	Major road
primary	National road
secondary	Regional road
tertiary	Local road
unclassified	Local access road
residential	Road in a residential area
living_street	Shared space street
pedestrian	Pedestrian street
track	Drivable track
footway	Pedestrian path
path	Non-drivable path
steps	Stairs
elevator	Elevator
corridor	Corridor
cycleway	Cycle path
road	Unknown classification

Remarks:

- For all red/yellow roads, _link can be added to indicate an access ramp.
- Traffic-related elements: highway=street_lamp, bus_stop, elevator, crossing, etc.

osmkey:public_transport

Table 2: Tag types for transport networks

Type	Element class	public_transport tag value
Platform	way	`platform`
Stop	point	`stop_position`
Station	way	`station`
Stop area	relation	`stop_area`

Since the `public_transport` tag is relatively recent, some infrastructures are still characterized by the keys `amenity`, `highway`, `railway`, etc.

3.4.3 Accessibility

osmkey:wheelchair The `wheelchair` key is used to report a place or path as suitable for wheelchair users.

Table 3: Common values for the `wheelchair` tag

Value	Description
yes	No access restrictions
limited	Partial access
no	No access
designated	Reserved for wheelchair users (rare)

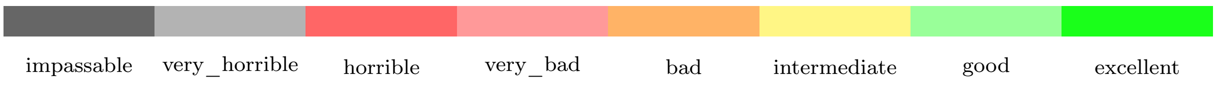
The subkeys `wheelchair:description` or `wheelchair:description:lang` can provide additional textual information.

osmkey:surface The `surface` key is used to describe the surface type of a road or path.

Table 4: Common values for the `surface` tag

Value	Description
paved	Paved (generic term)
unpaved	Unpaved (generic term)
asphalt	Asphalt
chipseal	Chip-sealed surface
concrete	Concrete
paving_stones	Regular paving stones
sett	Cut stone blocks
grass_paver	Mixture of grass and pavers
compacted	Compacted
gravel	Gravel
pebblestone	Pebbles
ground/dirt/earth	Earth
grass	Grass
sand	Sand

osmkey:smoothness The `smoothness` key is used to characterize the quality of the surface and is logically paired with the `surface` attribute to provide optimal inclusive routing. Values follow the scale below:



Observations :

- From `intermediate` level, the path is usable by motor vehicles and bicycles without significant speed reduction.
- Paths rated `very_bad` to `very_horrible` are only suitable for mountain bikes, 4x4s, or lifted vehicles in the best case.
- `impassable` paths are only accessible by pedestrians.

osmkey:kerb The `kerb` key is used on a node belonging to `highway=footway`, `highway=cycleway`, or `highway=path` where a curb is located. Values `no` and `flush` indicate a curb suitable for vehicle entry, while `lowered` indicates wheelchair accessibility.

osmkey:ramp The `ramp` key indicates the presence of an access ramp. It is commonly used with `highway=steps` and `highway=footway`. The typical value is `yes`. For wheelchair-standardized ramps, the subkey `ramp:wheelchair` can be used.

3.4.4 Safety

osmkey:lit The `lit` key indicates the presence of lighting (often on paths). Values can be boolean (`yes`/`no`) or more informative, e.g., `24/7` for continuous illumination, `limited` for lights installed but not always functional at night, or `automatic` for motion-detected lighting.

osmkey:surveillance The `surveillance` key indicates the presence of video surveillance cameras. Common values are `public`, `outdoor`, and `indoor`, though some objects use `yes`/`no`.

osmkey:maxspeed The `maxspeed` attribute indicates the maximum allowed speed on a path, in `km/h` if no unit is specified. If no speed limit applies, the value can be `none` or `no`. Implicit country-specific values may also be used: `maxspeed=<countrycode>:<zone type>`.

osmkey:covered The `covered` key indicates if the object is covered.

Table 5: Common values for the `covered` tag

Value	Description
yes	Fully covered (not including arcades or colonnades)
partial	Partially covered
no	Not covered
arcade	Covered by arches
colonnade	Covered by columns

3.4.5 Health

osmkey:emergency The `emergency` key is used to indicate the presence of emergency equipment and services, emergency rooms, or transport restrictions.

3.4.6 Access

osmkey:building The `building` key defines the outline of a building. Some useful values include:

Table 6: Common values for the `building` tag

Value	**Description**
apartments	Building mainly containing apartments
detached	Detached house
dormitory	University dormitory
farm	Farm
hotel	Hotel
house	House (generic)
residential	Residential building (generic)
terrace	Row of terraced houses
commercial	Building for tertiary sector activities
industrial	Building for industrial activities
office	Office
supermarket	Supermarket
toilets	Toilets
stadium	Stadium

Observations :

- There are over a hundred possible values for this tag; see the wiki for details.
- The simplest use is `building=yes`.
- The building's purpose (hospital, school, university, etc.) is usually indicated by the `amenity` key.

osmkey:access The `access` key describes legal restrictions for an element.

Table 7: Common values for the `access` tag

Value	**Description**
yes	Legal right of access
no	General prohibition
private	Access restricted to certain people
permissive	Public access allowed by owner
destination	Access only if the element is the destination (e.g., residents only)
delivery	Access reserved for deliveries
customers	Access reserved for customers

Observations :

- Physical vehicle restrictions are indicated by other keys (e.g., `maxheight`).
- Road-type restrictions are indicated with the `highway` key.

osmkey:opening_hours The `opening_hours` key indicates opening hours of the element. The syntax is relatively complex. The value typically defines hours day by day (or by day ranges), followed by time intervals. Each rule is separated by a semicolon and space (;).

Observations :

- `PH` indicates general rules for public holidays.
- `SH` indicates general rules for school holidays.
- `off` indicates closure for the entire day.
- The subkey `url` allows adding a link where hours can be checked.

osmkey:department On a university campus, it may be useful to know more about a building. The keys `department` and `faculty` provide information on the type of courses offered in a building.

4 Query set

Within the framework of use of our query engine (so the geographic area is always limited to within the Lyon 1 campus), we will focus particularly on the list of a certain type of element, filtered by a geospatial criterion (e.g., restaurants, university dormitories within a given radius, etc.) and by specific properties (e.g., accessibility for people with reduced mobility).

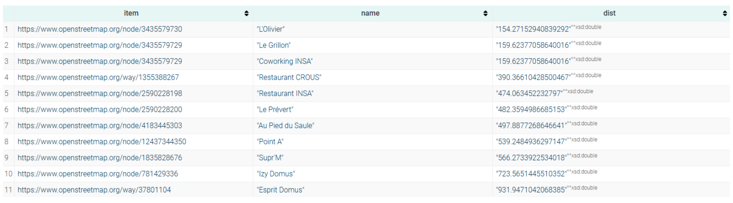
The objective in relation to the thesis is to propose routes adapted to various constraints (avoiding noisy roads, stairs, etc.) using diverse data sources (OSM, sensors, etc.). It's therefore necessary to filter the RDF graph in order to get the lightest subgraph possible containing the information relevant for route planning.

To make efficient route planning, parameterized SPARQL queries will be embed into JavaScript functions.

4.1 Urban data access

`getNearbyAmenities(endpointUrl, latUser, lonUser, amenity)`

- Get all amenities of a specific type and their distance to the user's coordinates.
- Returns an array of objects having the following structure: `(item, name, dist)`.
- Example of use:
  `getNearbyAmenities('http://localhost:7200/repositories/1', 45.784797, 4.876642, 'restaurant')`

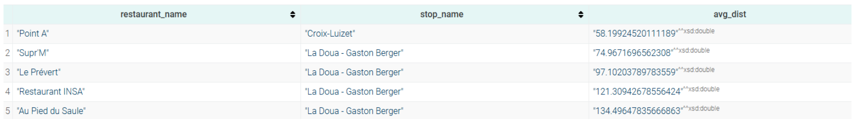


	item	name	dist
1	https://www.openstreetmap.org/node/3435579730	"L'Olivier"	"154.27152940839292"^^xsd:double
2	https://www.openstreetmap.org/node/3435579729	"Le Grillon"	"159.62377058640016"^^xsd:double
3	https://www.openstreetmap.org/node/3435579729	"Coworking INSA"	"159.62377058640016"^^xsd:double
4	https://www.openstreetmap.org/way/1355388267	"Restaurant CROUS"	"390.36610428500467"^^xsd:double
5	https://www.openstreetmap.org/node/2590228198	"Restaurant INSA"	"474.063452232797"^^xsd:double
6	https://www.openstreetmap.org/node/2590228200	"Le Prévert"	"482.3594986685153"^^xsd:double
7	https://www.openstreetmap.org/node/4183445303	"Au Pied du Saule"	"497.8877268646641"^^xsd:double
8	https://www.openstreetmap.org/node/12437344350	"Point A"	"539.2484936297147"^^xsd:double
9	https://www.openstreetmap.org/node/1835828676	"Supr'M"	"566.2733922534018"^^xsd:double
10	https://www.openstreetmap.org/node/781429336	"Izy Domus"	"723.5651445510352"^^xsd:double
11	https://www.openstreetmap.org/way/37801104	"Esprit Domus"	"931.9471042068385"^^xsd:double

Figure 5: Query results

`getAmenitiesNextToTransport(endpointUrl, amenity, distMax)`

- Get all amenities of a specific type located less than `distMax` meters from a public transport stop.
- Returns an array of objects having the following structure: `(item, item_name, stop, stop_name, dist)`.
- Example of use:
  `getAmenitiesNextToTransport('http://localhost:7200/repositories/1', 'restaurant', 200)`

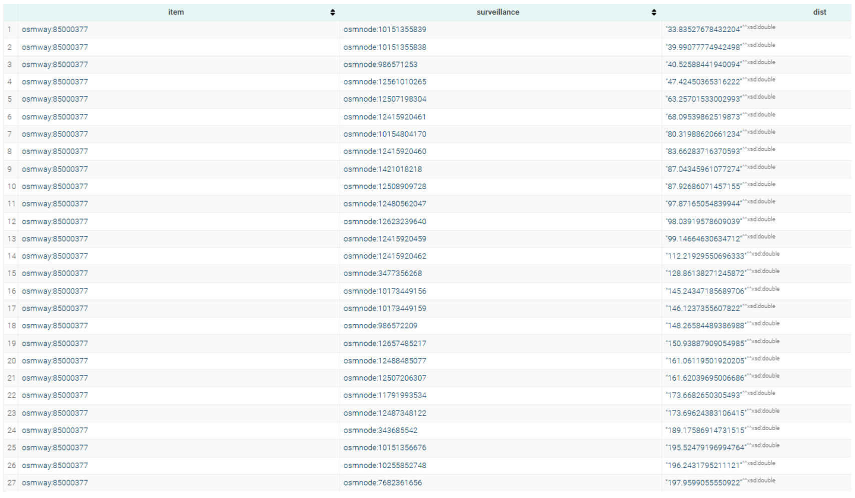


	restaurant_name	stop_name	avg_dist
1	"Point A"	"Croix-Luizet"	"58.19924520111189"^^xsd:double
2	"Supr'M"	"La Doua - Gaston Berger"	"74.9671696562308"^^xsd:double
3	"Le Prévert"	"La Doua - Gaston Berger"	"97.10203789783559"^^xsd:double
4	"Restaurant INSA"	"La Doua - Gaston Berger"	"121.30942678556424"^^xsd:double
5	"Au Pied du Saule"	"La Doua - Gaston Berger"	"134.49647835666863"^^xsd:double

Figure 6: Query results

`getNearbySurveillance(endpointUrl, id, distMax)`

- Get all surveillance cameras located less than `distMax` meters from a given OSM object.
- Returns an array of objects having the following structure: `(item, surveillance, dist)`.
- Example of use:
  `getNearbySurveillance('http://localhost:7200/repositories/1', 'osmway:85000377', 200)`

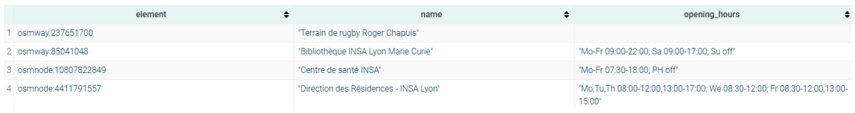


	item	surveillance	dist
1	osmway:85000377	osmnode:10151355839	"33.83527678432204"^^xsd:double
2	osmway:85000377	osmnode:10151355838	"39.99077774942498"^^xsd:double
3	osmway:85000377	osmnode:986571253	"40.52588441940094"^^xsd:double
4	osmway:85000377	osmnode:12561010265	"47.42450365316222"^^xsd:double
5	osmway:85000377	osmnode:12507198304	"63.25701533002993"^^xsd:double
6	osmway:85000377	osmnode:12415920461	"68.09539862519873"^^xsd:double
7	osmway:85000377	osmnode:10154804170	"80.31988620661234"^^xsd:double
8	osmway:85000377	osmnode:12415920460	"83.66283716370593"^^xsd:double
9	osmway:85000377	osmnode:1421018218	"87.04345961077274"^^xsd:double
10	osmway:85000377	osmnode:12508909728	"87.92686071457155"^^xsd:double
11	osmway:85000377	osmnode:12480562047	"97.87165054839944"^^xsd:double
12	osmway:85000377	osmnode:12623239640	"98.03919578609039"^^xsd:double
13	osmway:85000377	osmnode:12415920459	"99.14664630634712"^^xsd:double
14	osmway:85000377	osmnode:12415920462	"112.21929550696333"^^xsd:double
15	osmway:85000377	osmnode:3477356268	"128.86138271245872"^^xsd:double
16	osmway:85000377	osmnode:10173449156	"145.24347185689706"^^xsd:double
17	osmway:85000377	osmnode:10173449159	"146.1237355607822"^^xsd:double
18	osmway:85000377	osmnode:986572209	"148.26584489386988"^^xsd:double
19	osmway:85000377	osmnode:12657485217	"150.93887909054985"^^xsd:double
20	osmway:85000377	osmnode:12488485077	"161.06119501920205"^^xsd:double
21	osmway:85000377	osmnode:12507206307	"161.62039695006686"^^xsd:double
22	osmway:85000377	osmnode:11791993534	"173.6682650305493"^^xsd:double
23	osmway:85000377	osmnode:12487348122	"173.69624383106415"^^xsd:double
24	osmway:85000377	osmnode:343685542	"189.17586914731515"^^xsd:double
25	osmway:85000377	osmnode:10151356676	"195.52479196994764"^^xsd:double
26	osmway:85000377	osmnode:10255852748	"196.2431795211121"^^xsd:double
27	osmway:85000377	osmnode:7682361656	"197.9599055550922"^^xsd:double

Figure 7: Query results

`getOpeningTime(endpointUrl, buildings)`

- Get opening times for a list of buildings.
- Returns an array of objects having the following structure: `(item, name, opening_hours)`.
- Example of use:
  `getOpeningTime('http://localhost:7200/repositories/1', ["osmway:237651700", "osmway:85041048", "osmnode:10807822849", "osmnode:4411791557"])`

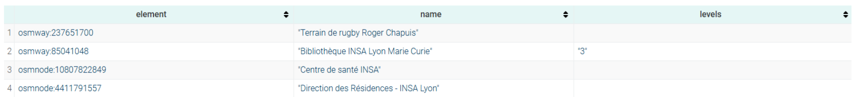


	element	name	opening_hours
1	osmway:237651700	"Terrain de rugby Roger Chapuis"	
2	osmway:85041048	"Bibliothèque INSA Lyon Marie Curie"	"Mo-Fr 09:00-22:00; Sa 09:00-17:00; Su off"
3	osmnode:10807822849	"Centre de santé INSA"	"Mo-Fr 07:30-18:00; PH off"
4	osmnode:4411791557	"Direction des Résidences - INSA Lyon"	"Mo,Tu,Th 08:00-12:00,13:00-17:00; We 08:30-12:00; Fr 08:30-12:00,13:00-15:00"

Figure 8: Query results

`getNumberOfLevels(endpointUrl, buildings)`

- Get the number of levels for a list of OSM buildings.
- Returns an array of objects having the following structure: `(item, name, levels)`.
- Example of use:
  `getNumberOfLevels('http://localhost:7200/repositories/1', ["osmway:237651700", "osmway:85041048", "osmnode:10807822849", "osmnode:4411791557"])`

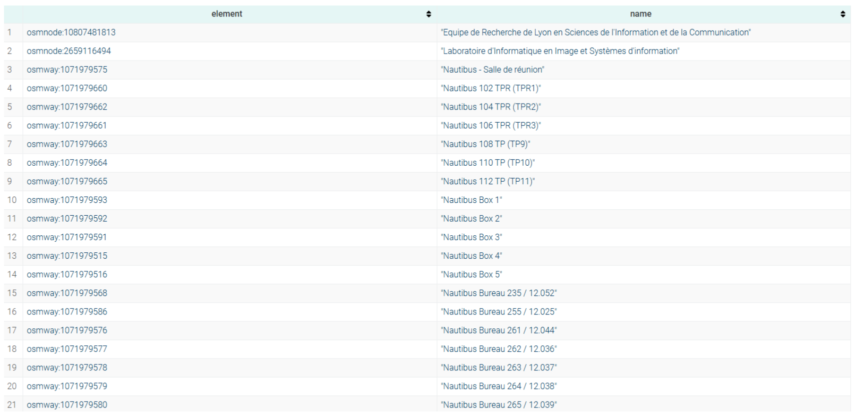


	element	name	levels
1	osmway:237651700	"Terrain de rugby Roger Chapuis"	
2	osmway:85041048	"Bibliothèque INSA Lyon Marie Curie"	"3"
3	osmnode:10807822849	"Centre de santé INSA"	
4	osmnode:4411791557	"Direction des Résidences - INSA Lyon"	

Figure 9: Query results

`getItemsInArea(endpointUrl, buildings)`

- Get all elements (nodes/ways) inside the area delimited by a list of building IDs.
- Returns an array of objects having the following structure: `(item, name)`.
- Example of use:
  `getItemsInArea('http://localhost:7200/repositories/1', ["osmway:237651700"])`



	element	name
1	osmnode:10807481813	"Equipe de Recherche de Lyon en Sciences de l'Information et de la Communication"
2	osmnode:2659116494	"Laboratoire d'Informatique en Image et Systèmes d'information"
3	osmway:1071979575	"Nautibus - Salle de réunion"
4	osmway:1071979660	"Nautibus 102 TPR (TPR1)"
5	osmway:1071979662	"Nautibus 104 TPR (TPR2)"
6	osmway:1071979661	"Nautibus 106 TPR (TPR3)"
7	osmway:1071979663	"Nautibus 108 TP (TP9)"
8	osmway:1071979664	"Nautibus 110 TP (TP10)"
9	osmway:1071979665	"Nautibus 112 TP (TP11)"
10	osmway:1071979593	"Nautibus Box 1"
11	osmway:1071979592	"Nautibus Box 2"
12	osmway:1071979591	"Nautibus Box 3"
13	osmway:1071979515	"Nautibus Box 4"
14	osmway:1071979516	"Nautibus Box 5"
15	osmway:1071979568	"Nautibus Bureau 235 / 12.052"
16	osmway:1071979586	"Nautibus Bureau 255 / 12.025"
17	osmway:1071979576	"Nautibus Bureau 261 / 12.044"
18	osmway:1071979577	"Nautibus Bureau 262 / 12.036"
19	osmway:1071979578	"Nautibus Bureau 263 / 12.037"
20	osmway:1071979579	"Nautibus Bureau 264 / 12.038"
21	osmway:1071979580	"Nautibus Bureau 265 / 12.039"

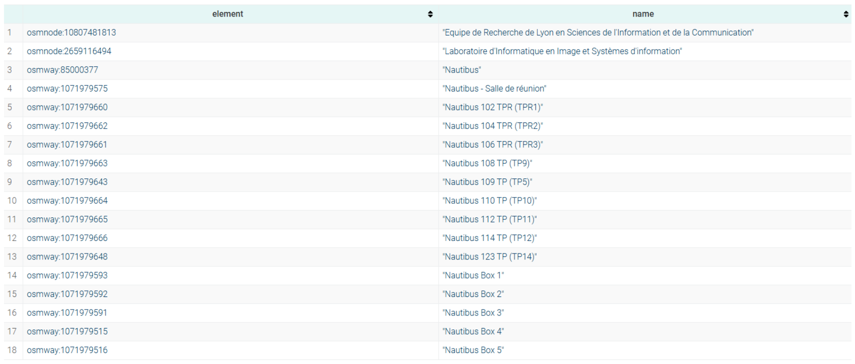
Figure 10: Query results

Total of 153 results

`getItemsInPolygon(endpointUrl, polygon)`

- Get all elements (nodes/ways) inside a custom polygon area.
- Returns an array of objects having the following structure: `(item, name)`.
- Example of use:

```
getItemsInPolygon('http://localhost:7200/repositories/1', "((4.8651301
45.7821498,4.8651635 45.782077,4.8651822 45.7820342,4.8651982
45.7820005,4.8652644 45.7820155,4.865272 45.7819992,4.8652966
45.7820041,4.8653646 45.7820199,4.8654831 45.7820464,4.8655199
45.7820558,4.8656041 45.7820735,4.8656725 45.7820888,4.8657436
45.7821047,4.865736 45.7821209,4.8657581 45.7821267,4.8658042
45.7821376,4.8658478 45.782147,4.8660656 45.7821958,4.8661089
45.7822054,4.8661509 45.7822144,4.8661912 45.7822231,4.8661656
45.7822793,4.8661521 45.7823084,4.8661513 45.782311,4.8661454
45.7823235,4.8661233 45.7823716,4.8660566 45.782356,4.8660489
45.7823553,4.8660424 45.7823535,4.8660086 45.7823463,4.8659923
45.7823419,4.8659679 45.7823372,4.8659626 45.7823357,4.86593
45.7823282,4.865928 45.7823283,4.865889 45.7823197,4.8658656
45.782313,4.8657977 45.7822991,4.865564 45.7822467,4.865476
45.7822268,4.8655024 45.7822332,4.8654528 45.7822216,4.8654089
45.7822123,4.8653011 45.7821881,4.8652591 45.7821785,4.8651301
45.7821498))")
```



	element	name
1	osmnode:10807481813	"Equipe de Recherche de Lyon en Sciences de l'Information et de la Communication"
2	osmnode:2659116494	"Laboratoire d'Informatique en Image et Systèmes d'information"
3	osmway:85000377	"Nautibus"
4	osmway:1071979575	"Nautibus - Salle de réunion"
5	osmway:1071979660	"Nautibus 102 TPR (TPR1)"
6	osmway:1071979662	"Nautibus 104 TPR (TPR2)"
7	osmway:1071979661	"Nautibus 106 TPR (TPR3)"
8	osmway:1071979663	"Nautibus 108 TP (TP9)"
9	osmway:1071979643	"Nautibus 109 TP (TP5)"
10	osmway:1071979664	"Nautibus 110 TP (TP10)"
11	osmway:1071979665	"Nautibus 112 TP (TP11)"
12	osmway:1071979666	"Nautibus 114 TP (TP12)"
13	osmway:1071979648	"Nautibus 123 TP (TP14)"
14	osmway:1071979593	"Nautibus Box 1"
15	osmway:1071979592	"Nautibus Box 2"
16	osmway:1071979591	"Nautibus Box 3"
17	osmway:1071979515	"Nautibus Box 4"
18	osmway:1071979516	"Nautibus Box 5"

Figure 11: Query results

Total of 149 results

Listing 1: SPARQL query to find node–road pairs within a zone close to the start and end nodes.

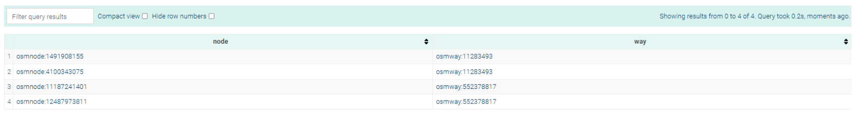
```
SELECT DISTINCT ?node ?way WHERE {
  BIND (osmnode:100218028 AS ?start) #start node
  BIND (osmnode:3868053587 AS ?end) #end node

  ?start geo:hasGeometry ?geo_start .
  ?geo_start geo:asWKT ?wkt_start .
  ?end geo:hasGeometry ?geo_end .
  ?geo_end geo:asWKT ?wkt_end .

  ?node rdf:type osm:node .
  ?node geo:hasGeometry/geo:asWKT ?wkt_node .
```

```
    ?way rdf:type osm:way .
    ?way osmkey:highway ?highway .
    FILTER (?highway IN ('residential', 'primary', 'secondary', '
       tertiary', 'unclassified'))
    ?way ogc:sfContains ?node .

    BIND (geof:distance(?wkt_start,?wkt_end, uom:metre) AS ?totalDist)
    BIND (geof:distance(?wkt_start,?wkt_node, uom:metre) AS ?distStart
       )
    BIND (geof:distance(?wkt_end,?wkt_node, uom:metre) AS ?distEnd)
    FILTER(?distStart + ?distEnd <= ?totalDist * 1.5)
  }
  LIMIT 100
```


Filter query results   Compact view ☐ Hide row numbers ☐   Showing results from 0 to 4 of 4. Query took 0.2s, moments ago.

	node	way
1	osmnode:1491908155	osmway:11283493
2	osmnode:4100343075	osmway:11283493
3	osmnode:11187241401	osmway:552378817
4	osmnode:12487973811	osmway:552378817

Figure 12: Query results

It is possible to use `?way osmkey:covered 'yes'` or `?way osmkey:lit 'yes'` in order to have only covered or enlightened paths.

4.2 Proximity analysis

`isCrossingNearby(endpointUrl, user_loc, distMax)`

- Returns a boolean: `true` if there is a crossing or footway crossing within `distMax` meters of the user (POINT geometry).
- Example of use:
  `isCrossingNearby('http://localhost:7200/repositories/1', '(4.8766485 45.7847625)'`
  `50)`

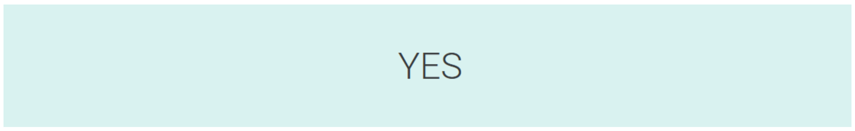
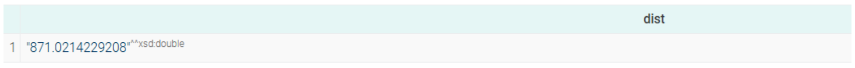

YES

Figure 13: Query results

`getDistance(endpointUrl, user_loc, item)`

- Get the distance in meters from a user's location (POINT) to a given object.
- Returns a float.
- Example of use:
  `getDistance('http://localhost:7200/repositories/1',`
  `'(4.8766485 45.7847625)', "osmnode:10807822849")`



	dist
1	"871.0214229208"^^xsd:double

Figure 14: Query results

`isAmenityOnPath(endpointUrl, path, amenity, distMax = 0)`

- Returns a boolean: `true` if an amenity is located on or near (within `distMax` meters) a given path (LINESTRING geometry).
- Examples of use:
  ```
  isAmenityOnPath('http://localhost:7200/repositories/1',
  '(4.8712942 45.7825353,4.8712781 45.7825318,4.8712394 45.7825236,4.8711957
  45.7825205,4.8711139 45.7825148,4.8710693 45.7825064,4.87068
  45.7824329,4.8702728 45.7823399,4.87025 45.7823347,4.8702336
  45.7823307,4.8700828 45.7822942,4.8700371 45.7822766,4.8699845
  45.7822395,4.8699645 45.7822,4.8699542 45.7821331,4.869959
  45.7820394,4.8699561 45.7820226,4.8699443 45.7819661,4.8698961
  45.7818372,4.8698187 45.7817759,4.8697547 45.7817252)', 'toilets')
  ```
  or
  ```
  isAmenityOnPath('http://localhost:7200/repositories/1',
  '(4.8712942 45.7825353,4.8712781 45.7825318,4.8712394 45.7825236,4.8711957
  45.7825205,4.8711139 45.7825148,4.8710693 45.7825064,4.87068
  45.7824329,4.8702728 45.7823399,4.87025 45.7823347,4.8702336
  45.7823307,4.8700828 45.7822942,4.8700371 45.7822766,4.8699845
  45.7822395,4.8699645 45.7822,4.8699542 45.7821331,4.869959
  45.7820394,4.8699561 45.7820226,4.8699443 45.7819661,4.8698961
  45.7818372,4.8698187 45.7817759,4.8697547 45.7817252)', 'toilets', 50)
  ```

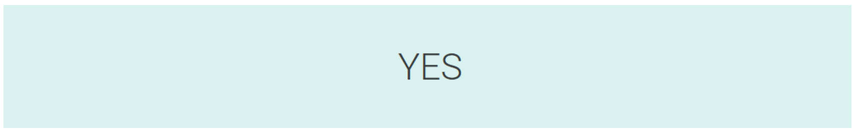
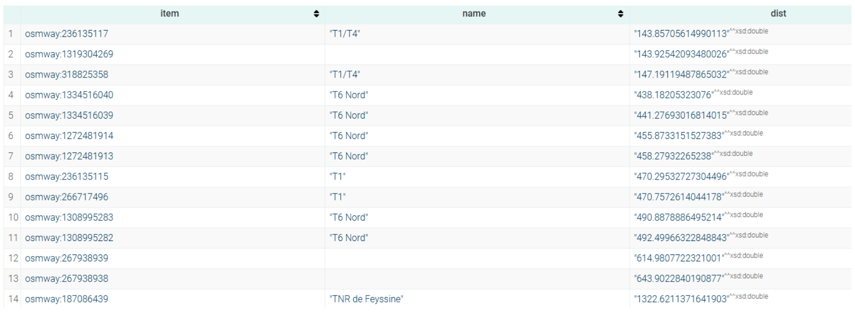

YES

Figure 15: Query results

`getDistanceToRailway(endpointUrl, geometry, geometryType, railwayType)`

- Get the minimal distance from a geometry to each railway of a specific type.
- Returns an array of objects having the following structure: `(item, name, dist)`. Name can be null if undefined in OSM.
- Example of use:
  ```
  getDistanceToRailway('http://localhost:7200/repositories/1',
  '((4.8651301 45.7821498,4.8651635 45.782077,4.8651822 45.7820342,4.8651982
  45.7820005,4.8652644 45.7820155,4.865272 45.7819992,4.8652966
  45.7820041,4.8653646 45.7820199,4.8654831 45.7820464,4.8655199
  45.7820558,4.8656041 45.7820735,4.8656725 45.7820888,4.8657436
  45.7821047,4.865736 45.7821209,4.8657581 45.7821267,4.8658042
  45.7821376,4.8658478 45.782147,4.8660656 45.7821958,4.8661089
  45.7822054,4.8661509 45.7822144,4.8661912 45.7822231,4.8661656
  45.7822793,4.8661521 45.7823084,4.8661513 45.782311,4.8661454
  45.7823235,4.8661233 45.7823716,4.8660566 45.782356,4.8660489
  45.7823553,4.8660424 45.7823535,4.8660086 45.7823463,4.8659923
  45.7823419,4.8659679 45.7823372,4.8659626 45.7823357,4.86593
  45.7823282,4.865928 45.7823283,4.865889 45.7823197,4.8658656
  ```

```
45.782313,4.8657977 45.7822991,4.865564 45.7822467,4.865476
45.7822268,4.8655024 45.7822332,4.8654528 45.7822216,4.8654089
45.7822123,4.8653011 45.7821881,4.8652591 45.7821785,4.8651301
45.7821498))', 'POLYGON', 'tram')
```



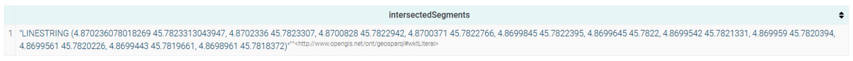
	item	name	dist
1	osmway:236135117	"T1/T4"	"143.85705614990113"^^xsd:double
2	osmway:1319304269		"143.92542093480026"^^xsd:double
3	osmway:318825358	"T1/T4"	"147.19119487865032"^^xsd:double
4	osmway:1334516040	"T6 Nord"	"438.18205323076"^^xsd:double
5	osmway:1334516039	"T6 Nord"	"441.27693016814015"^^xsd:double
6	osmway:1272481914	"T6 Nord"	"455.8733151527383"^^xsd:double
7	osmway:1272481913	"T6 Nord"	"458.27932265238"^^xsd:double
8	osmway:236135115	"T1"	"470.29532727304496"^^xsd:double
9	osmway:266717496	"T1"	"470.7572614044178"^^xsd:double
10	osmway:1308995283	"T6 Nord"	"490.8878886495214"^^xsd:double
11	osmway:1308995282	"T6 Nord"	"492.49966322848843"^^xsd:double
12	osmway:267938939		"614.9807722321001"^^xsd:double
13	osmway:267938938		"643.9022840190877"^^xsd:double
14	osmway:187086439	"TNR de Feyssine"	"1322.6211371641903"^^xsd:double

Figure 16: Query results

`getIntersection(endpointUrl, path, area)`

- Get the intersection between a path and an area.
- Returns a LINESTRING.
- Example of use:

```
getIntersection('http://localhost:7200/repositories/1',
'(4.8712942 45.7825353,4.8712781 45.7825318,4.8712394 45.7825236,4.8711957
45.7825205,4.8711139 45.7825148,4.8710693 45.7825064,4.87068
45.7824329,4.8702728 45.7823399,4.87025 45.7823347,4.8702336
45.7823307,4.8700828 45.7822942,4.8700371 45.7822766,4.8699845
45.7822395,4.8699645 45.7822,4.8699542 45.7821331,4.869959
45.7820394,4.8699561 45.7820226,4.8699443 45.7819661,4.8698961
45.7818372,4.8698187 45.7817759,4.8697547 45.7817252)',
'((4.8683676 45.7824128,4.8683821 45.7823836,4.8683946 45.7823582,4.8687298
45.7816229,4.8687445 45.7815799,4.8689635 45.7816277,4.8696959
45.781794,4.869726 45.7818008,4.8697488 45.781806,4.869768
45.7818104,4.8697922 45.781816,4.8698689 45.7818312,4.8698961
45.7818372,4.8699128 45.7818412,4.8700201 45.7818601,4.8701484
45.78191,4.8702211 45.7819146,4.8701499 45.7820892,4.8701061
45.7821967,4.8702853 45.7822363,4.8702408 45.782319,4.8702304
45.7823461,4.8702137 45.7823883,4.8702051 45.782406,4.8701198
45.7825933,4.8700822 45.7825955,4.8700515 45.7826659,4.8700289
45.7827169,4.8700045 45.7827679,4.869995 45.7827911,4.8699767
45.7828308,4.869796 45.7827905,4.8697921 45.782807,4.8688195
45.7825926,4.8687668 45.7825184,4.8683676 45.7824128))')
```



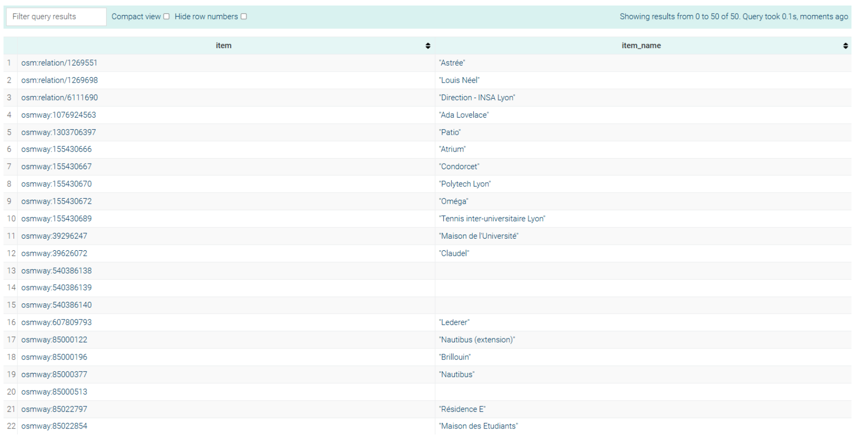
	intersectedSegments
1	"LINESTRING (4.870236078018269 45.7823313043947, 4.8702336 45.7823307, 4.8700828 45.7822942, 4.8700371 45.7822766, 4.8699845 45.7822395, 4.8699645 45.7822, 4.8699542 45.7821331, 4.869959 45.7820394, 4.8699561 45.7820226, 4.8699443 45.7819661, 4.8698961 45.7818372)"^^<http://www.opengis.net/ont/geosparql#wktLiteral>

Figure 17: Query results

4.3 Accessibility and mobility

`getWheelchairFriendlyBuildings(endpointUrl)`

- Get all buildings that are wheelchair-accessible.
- Returns an array of objects having the following structure: `(item, name)`. Name can be null if undefined in OSM.
- Example of use:
  `getWheelchairFriendlyBuildings('http://localhost:7200/repositories/1')`



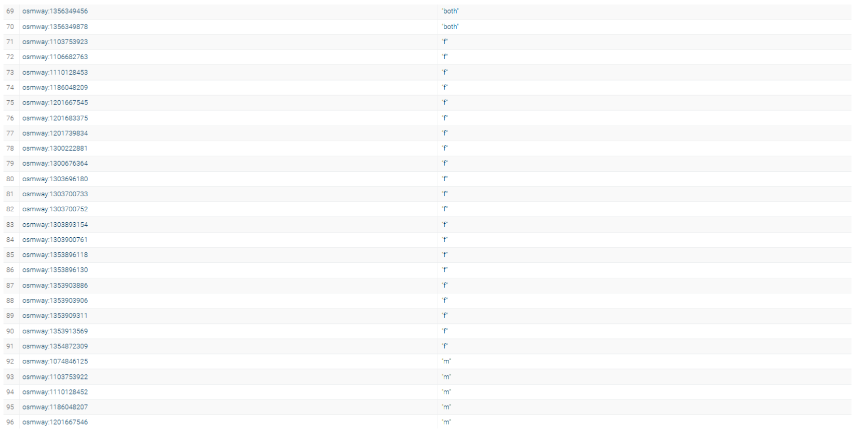
Filter query results | Compact view ☐ Hide row numbers ☐ | Showing results from 0 to 50 of 50. Query took 0.1s, moments ago

	item	item_name
1	osm:relation/1269551	"Astrée"
2	osm:relation/1269698	"Louis Néel"
3	osm:relation/6111690	"Direction - INSA Lyon"
4	osmway:1076924563	"Ada Lovelace"
5	osmway:1303706397	"Patio"
6	osmway:155430666	"Atrium"
7	osmway:155430667	"Condorcet"
8	osmway:155430670	"Polytech Lyon"
9	osmway:155430672	"Oméga"
10	osmway:155430689	"Tennis inter-universitaire Lyon"
11	osmway:39296247	"Maison de l'Université"
12	osmway:39626072	"Claudel"
13	osmway:540386138	
14	osmway:540386139	
15	osmway:540386140	
16	osmway:607809793	"Lederer"
17	osmway:85000122	"Nautibus (extension)"
18	osmway:85000196	"Brillouin"
19	osmway:85000377	"Nautibus"
20	osmway:85000513	
21	osmway:85022797	"Résidence E"
22	osmway:85022854	"Maison des Etudiants"

Figure 18: Query results

`getWheelchairFriendlyToilets(endpointUrl)`

- Get all toilets tagged as wheelchair-accessible (with info about gender, if available).
- Returns an array of objects having the following structure: `(item, genre)`. Genre can be null if undefined in OSM.
- Example of use:
  `getWheelchairFriendlyToilets('http://localhost:7200/repositories/1')`



69	osmway:1356349456	"both"
70	osmway:1356349878	"both"
71	osmway:1103753923	"f"
72	osmway:1106682763	"f"
73	osmway:1110128453	"f"
74	osmway:1186048209	"f"
75	osmway:1201667545	"f"
76	osmway:1201683375	"f"
77	osmway:1201739834	"f"
78	osmway:1300222881	"f"
79	osmway:1300676364	"f"
80	osmway:1303696180	"f"
81	osmway:1303700733	"f"
82	osmway:1303700752	"f"
83	osmway:1303893154	"f"
84	osmway:1303900761	"f"
85	osmway:1353896118	"f"
86	osmway:1353896130	"f"
87	osmway:1353903886	"f"
88	osmway:1353903906	"f"
89	osmway:1353909311	"f"
90	osmway:1353913569	"f"
91	osmway:1354872309	"f"
92	osmway:1074846125	"m"
93	osmway:1103753922	"m"
94	osmway:1110128452	"m"
95	osmway:1186048207	"m"
96	osmway:1201667546	"m"

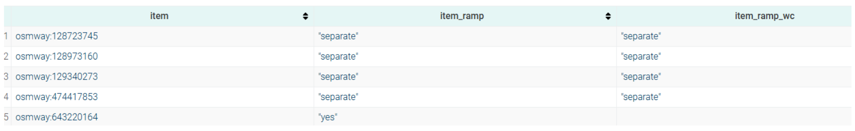
Figure 19: Query results

Total of 116 results

`getStairsWithRamp(endpointUrl)`

- Get the list of stairs equipped with a ramp.

- Returns an array of objects having the following structure: `(item, item_ramp, wheelchair_accessible_ramp)`.
- Example of use:
`getStairsWithRamp('http://localhost:7200/repositories/1')`

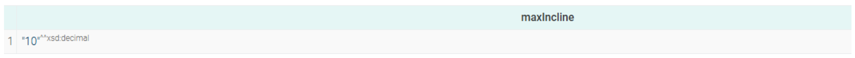


	item	item_ramp	item_ramp_wc
1	osmway:128723745	"separate"	"separate"
2	osmway:128973160	"separate"	"separate"
3	osmway:129340273	"separate"	"separate"
4	osmway:474417853	"separate"	"separate"
5	osmway:643220164	"yes"	

Figure 20: Query results

`getMaximumGradient(endpointUrl, path)`

- Get the maximum gradient (in %) along a path defined by a list of OSM segment IDs.
- Returns a float.
- Example of use:
`getMaximumGradient('http://localhost:7200/repositories/1',["osmway:43013911", "osmway:1113003348", "osmway:128973167", "osmway:1113003347", "osmway:643215532", "osmway:1342103505", "osmway:1342103503"])`

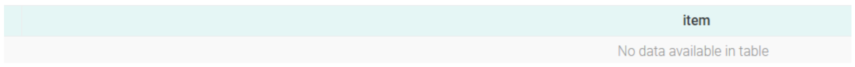


	maxIncline
1	"10"^^xsd:decimal

Figure 21: Query results

`getFloorTransitions(endpointUrl, building)`

- Get the elements (nodes and segments) that model vertical connections (stairs, elevators, ramps) in a building.
- Returns an array of objects having the following structure: `(item, highway, level)`.
- Example of use:
`getFloorTransitions('http://localhost:7200/repositories/1', 'osmway:85000377')`



item
No data available in table

Figure 22: Query results

Only `corridor` are mapped inside the building, so there is no result for steps or lifts

5 OpenMeteo

A request example is `https://api.open-meteo.com/v1/forecast?latitude=45.7843461&longitude=4.8762278&daily=uv_index_max&hourly=temperature_2m,rain,snowfall,precipitation_probability,precipitation,relative_humidity_2m,wind_speed_10m,wind_direction_10m,visibility&timezone=auto&forecast_days=3`.

Interesting data could be `temperature_2m`, `rain`, `snowfall`, `precipitation_probability`, `precipitation`, `relative_humidity_2m`, `wind_speed_10m`, `wind_direction_10m`, `visibility`.

Given resolution and campus size, the campus can be split into two areas :
The first is defined by `POLYGON ((4.8704149 45.784944, 4.8700209 45.7848784, 4.8697659 45.7848368, 4.8692491 45.7847672, 4.8686879 45.7846952, 4.8684059 45.7846297, 4.8681377 45.7846084, 4.8665963 45.7844707, 4.8659784 45.7844435, 4.8652228 45.7843653, 4.8650107 45.7843177, 4.8648152 45.7843348, 4.8645738 45.7843399, 4.8643927 45.7843231, 4.8640849 45.7843212, 4.8638234 45.7843301, 4.8637034 45.7843148, 4.8634105 45.7842439, 4.8632561 45.7842001, 4.8631211 45.7841386, 4.8631218 45.784123, 4.8630376 45.7841225, 4.8630409 45.7841547, 4.862912 45.7841421, 4.8627924 45.78417, 4.8625345 45.7841236, 4.8621394 45.7839768, 4.8620483 45.783902, 4.8619514 45.7838316, 4.8618766 45.7837589, 4.8614705 45.7833643, 4.8610846 45.782956, 4.861506 45.7827553, 4.8612067 45.7823572, 4.8608597 45.7816941, 4.8607484 45.7813434, 4.8612932 45.7810899, 4.861408 45.7810742, 4.8614227 45.7810719, 4.8614771 45.7810591, 4.8615439 45.7810204, 4.8615789 45.780957, 4.8618847 45.7808147, 4.8619406 45.7807887, 4.8620543 45.780577, 4.8621126 45.7804684, 4.8619016 45.7802738, 4.8614247 45.7798341, 4.8612243 45.7796252, 4.8614776 45.7796019, 4.8615287 45.7795973, 4.8623204 45.7795196, 4.8625067 45.7795075, 4.8628471 45.7794759, 4.8629911 45.7794626, 4.8651913 45.7792549, 4.8654151 45.7792338, 4.8662363 45.7791522, 4.8664418 45.7791318, 4.8667688 45.7791087, 4.8671426 45.7790822, 4.8679866 45.7790375, 4.8686123 45.7790173, 4.8697581 45.7790127, 4.872422 45.779145)).`
having `4.8662 45.7821` as centroid.
The other one is remaining part of the campus, corresponding to `POLYGON((4.872422 45.779145, 4.8725172 45.7792446, 4.8726129 45.7792564, 4.8731304 45.7793323, 4.8732642 45.7793527, 4.8735868 45.7794082, 4.8738174 45.7794502, 4.8739439 45.7794732, 4.8742883 45.7795413, 4.8746986 45.7796304, 4.8747558 45.7796417, 4.8751384 45.7797337, 4.8755028 45.780427, 4.8755361 45.7804893, 4.875836 45.7810509, 4.8758721 45.7811185, 4.8758951 45.7811583, 4.8759312 45.7812267, 4.876005 45.7813677, 4.8766141 45.782141, 4.8766379 45.7821713, 4.8766775 45.7822031, 4.8794125 45.7828106, 4.8798242 45.7828834, 4.8832591 45.7836541, 4.8833331 45.7836707, 4.8833391 45.7836721, 4.8836979 45.7837526, 4.8837873 45.7838084, 4.8846355 45.7839943, 4.8846245 45.7840169, 4.8846161 45.7840356, 4.8845549 45.7840397, 4.8844948 45.7840497, 4.8844513 45.7840727, 4.8843964 45.7841236, 4.8843203 45.7842064, 4.883753 45.7854721, 4.8854094 45.786398, 4.8849511 45.7871588, 4.884296 45.787262, 4.883883 45.7873139, 4.8834015 45.787349, 4.8824781 45.7873448, 4.8822106 45.7873345, 4.88188 45.7872936, 4.8817125 45.7872686, 4.8814071 45.7872229, 4.8811971 45.7871915, 4.8808388 45.7871359, 4.8806113 45.7870909, 4.8792326 45.7867852, 4.8790018 45.7867329, 4.8789531 45.7867228, 4.878895 45.7867102, 4.878154 45.7865451, 4.8780953 45.786532, 4.8766212 45.7862038, 4.8755058 45.7859421, 4.8747776 45.785781, 4.8731311 45.7854332, 4.8717737 45.7851681, 4.8712329 45.7850789, 4.8704149 45.784944, 4.8708837 45.7832868, 4.8709694 45.7831057, 4.8710306 45.7829774, 4.8710514 45.7829315, 4.8712394 45.7825236, 4.8713595 45.782263, 4.8713848 45.7822081, 4.8714025 45.7821697, 4.8716349 45.7816655, 4.8716481 45.7816371, 4.8716645 45.7816019, 4.8716743 45.7815809, 4.8716876 45.781553, 4.871801 45.7813058, 4.8718167 45.7812728, 4.8718316 45.7812419, 4.871873 45.7811515, 4.872422 45.779145))`
having `4.8772 45.7833` as centroid.

Those ways could get an `hasMeteo` attribute linking them to a meteo item that provides an inventory of weather informations.

The maximum number of API requests is 10 000 per day, 5 000 per hour and 600 per minute.

Even if it's practically possible to do requests directly on the API and extract JSON files, we could convert API results into RDF data to keep an homogeneity and stay in Web Semantic with SPARQL Micro-Services.