

An aerial photograph of a winding asphalt road through a dense green forest. Several cars are visible on the road, including a white car, a blue car, and a dark car. The road curves through the trees, and the overall scene is captured from a high angle.

# Overpass: oltre il wizard

A CRASH COURSE

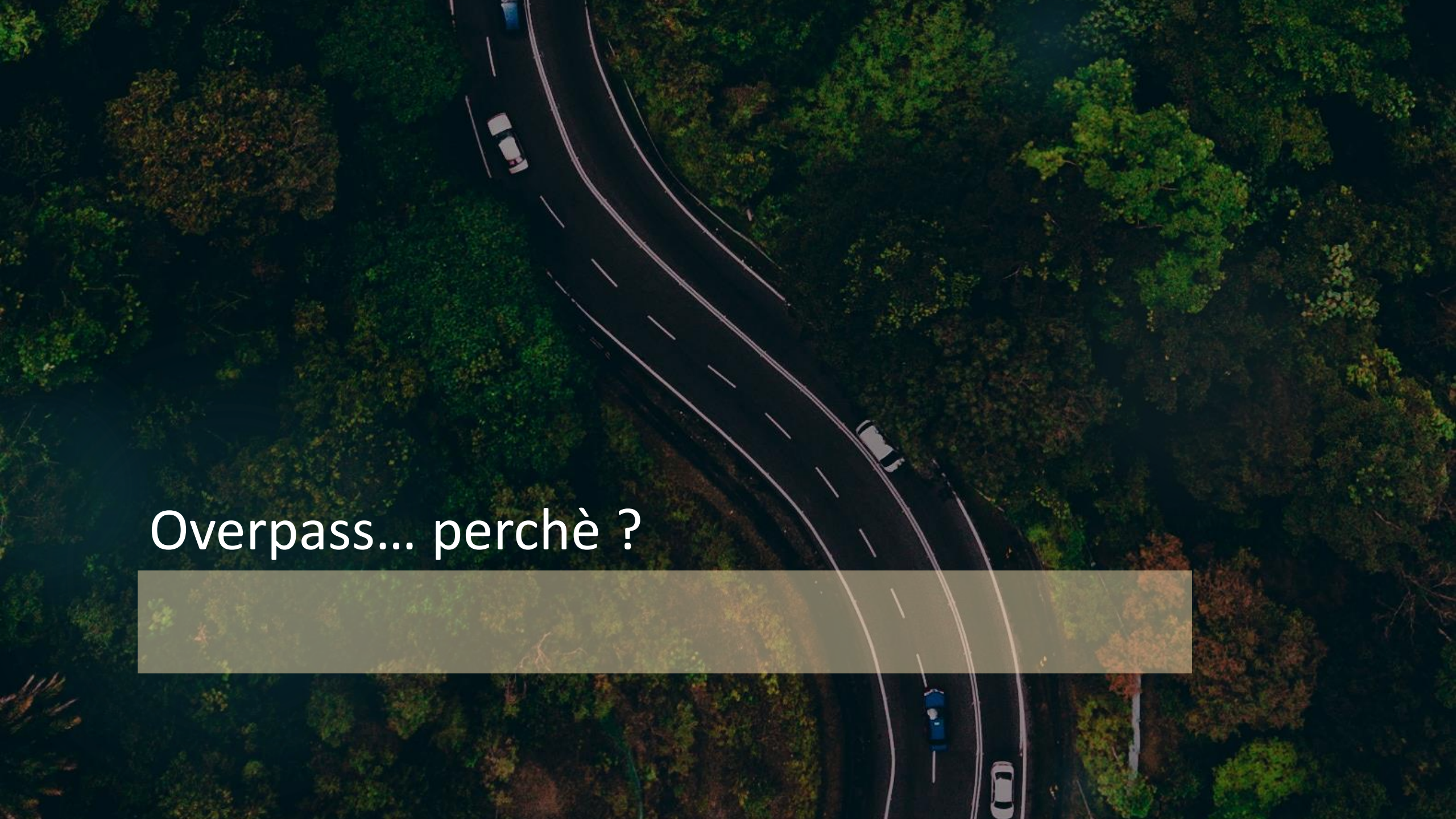
Relatore: Andrea Albani (IlBano on OSM)

# Obiettivi

- ▶ Identikit di Overpass-API
- ▶ Capire Overpass QL
- ▶ Costruire una query complessa in Overpass QL

# Pre-requisiti

- ▶ Ho una qualche conoscenza del modello dati di OSM (nodi, way, relazioni, tags)
- ▶ Ho già usato Overpass Turbo e il suo wizard

An aerial photograph of a winding asphalt road cutting through a dense, lush green forest. The road curves from the top left towards the bottom right. Several cars are visible on the road: a white car in the upper left, a white car in the middle, a blue car in the lower right, and another white car at the very bottom. The forest is thick with trees, and the lighting suggests a slightly overcast or late afternoon setting. A semi-transparent horizontal bar is overlaid across the middle of the image, containing the text.

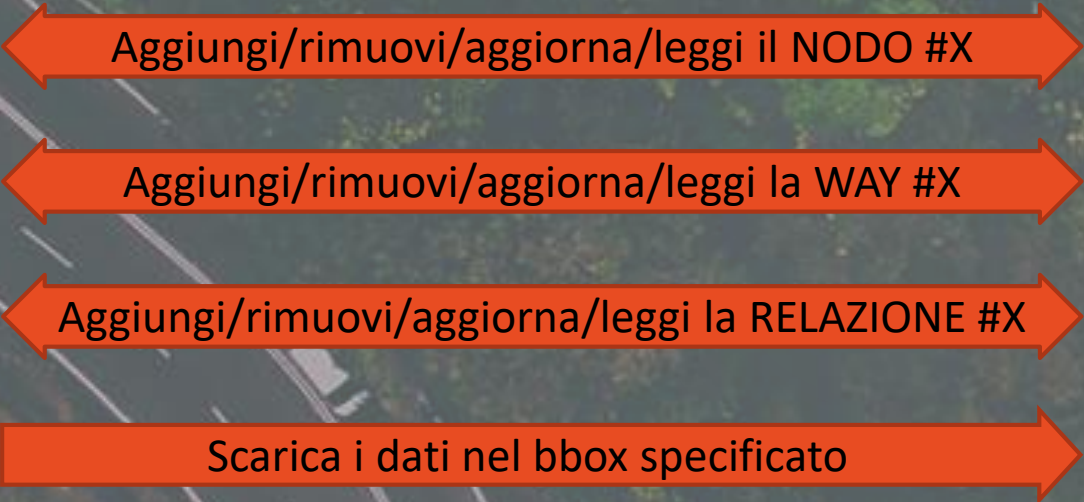
Overpass... perchè ?

# I dati in OSM



- ▶ Nodi
- ▶ Way
- ▶ Relazioni
- .... e loro tags

OSM API V 0.6



Data consumer  
(es. JOSM)

Limiti -> Posso leggere un dato solo se ne conosco il suo OSM #id (!)

# 2011: arriva Overpass-API



Limiti -> ?

# I linguaggi di Overpass API

Le Overpass API accettano due tipi di linguaggio:

- ▶ Overpass XML: basato sulla sintassi XML. Più leggibile, ma richiede molti caratteri per codificare una richiesta
- ▶ Overpass QL: introdotto in un secondo momento e diventato poi di fatto il linguaggio standard. E' molto conciso e anche di non immediata comprensione

Esempio: la medesima query nei due linguaggi

Overpass XML	Overpass QL
<pre>&lt;union&gt;   &lt;bbox-query s="51.249" w="7.148" n="51.251" e="7.152"/&gt;   &lt;recurse type="up"/&gt; &lt;/union&gt; &lt;print mode="meta"/&gt;</pre>	<pre>( node (51.249,7.148,51.251,7.152) ; &lt; ; ); out meta;</pre>

# Overpass-API: l'autore



Roland a SOTM 2019

- ▶ **Roland Olbricht** – Informatico
- ▶ Si pone un obiettivo... “fornire dati ai tool” che trattano dati OSM
- ▶ Sviluppa Overpass-API con i seguenti requisiti:
  - ❑ Linguaggio di interrogazione versatile ed estensibile
  - ❑ Risposte rapide
  - ❑ Requisiti hardware ragionevoli
  - ❑ Installazione semplice / poca manutenzione
- ▶ Mantiene tuttora attivo il servizio



Select a/the relation(s) with  
network=?, ref=?  
- and the corresponding nodes  
that are relevant  
for public transport  
- and all relations referring  
to these nodes.



```
<query type="relation">  
  <has-kv k="network" v="?"/>  
  <has-kv k="ref" v="?"/>  
</query>  
<recurse type="relation-node" into="stops"/>  
<print from="stops"/>  
<union>  
  <query type="node"><around from="stops"/>  
  <has-kv k="highway" v="bus_stop"/>  
</query>  
  <query type="node"><around from="stops"/>  
  <has-kv k="railway"/>  
</query>  
</union>  
<print/>  
<recurse type="node-relation"/>  
<print/>
```





An aerial photograph of a winding asphalt road through a dense green forest. The road curves from the top left towards the bottom right. Several cars are visible on the road: a white car in the upper left, a white car in the middle, a blue car in the lower right, and another white car at the bottom. A semi-transparent horizontal bar is overlaid across the middle of the image, containing the text 'Overpass QL'.

# Overpass QL

# Elementi base

OverpassQL è il linguaggio usato per interfacciarsi con Overpass API

```
[out:xml] [timeout:25] ;
```

```
(  
  way["leisure"="park"] ({{bbox}}) ;  
  way["amenity"] ({{bbox}}) ;  
) ->.MieiOggetti ;
```

```
out body;
```

```
>;
```

```
out skel qt;
```

**Settings:** impostazioni che controllano il funzionamento di Overpass

**Statements:** sono le istruzioni per manipolare i dati. Leggono e scrivono le informazioni in un set. Se il set in input o output non è specificato usano il default set `._`

**Set:** variabili che memorizzano le informazioni

**Block statements:** raggruppano statements per consentire operazioni sul loro insieme

An aerial photograph of a winding asphalt road cutting through a dense, lush green forest. The road curves from the top left towards the bottom right. Several cars are visible on the road: a white car in the upper left, a white car in the middle, a blue car in the lower right, and another white car at the very bottom. The forest is thick with trees, and the lighting suggests a bright, sunny day.

# Settings

# Settings

I settings, il cui inserimento è opzionale, servono a definire:

- ▶ il formato dei dati in output

`[out:xml]` formato OSM (default)

`[out:json]` formato json

`[out:csv....]` formato tabellare CSV

- ▶ il tempo massimo di esecuzione

`[timeout:secondi]` se non viene specificato è 180

- ▶ la quantità di memoria utilizzata dalla query

`[maxsize:bytes]` il default è 512 MB. Serve per query complesse, ma non vi è certezza che i server di Overpass API la riservino

- ▶ un bounding box globale

`[bbox:lat sud,lon ovest,lat nord,lon est]`

- ▶ dei periodi di tempo in cui esaminare i dati:

`[date:data]` `[diff:data1<,data2>]` `[adiff:data1<,data2>]`

I settings, concatenati, devono essere messi nella prima riga e devono terminare con `;` es:

```
[out:xml] [timeout:60] [bbox:45.20,10.71,45.30,10.88];
```

An aerial photograph of a winding asphalt road cutting through a dense, lush green forest. The road curves from the top left towards the bottom right. Several cars are visible on the road: a white car in the upper left, a white car in the middle, a blue car in the lower right, and another white car at the very bottom. The forest is thick with trees, and the lighting suggests a bright day. A semi-transparent horizontal bar is overlaid across the middle of the image.

# Statements

# Statements: query base

Gli statement più importanti sono detti di query e servono per estrarre i dati da OSM:

`node`

`way`

`rel`

Sono spesso seguiti da un filtro basato sui tag OSM:

```
rel[name="Parco Mangoni"] ;
```

```
node["name:en"]="Turin" ;
```

Se il nome o il valore del tag contiene caratteri diversi da quelli dell'alfabeto, bisogna usare gli apici singoli o doppi

I filtri possono essere concatenati

```
way[leisure=park][name="Parco Mangoni"] ;
```

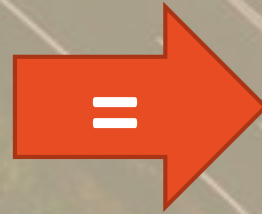
In questo esempio verranno estratte le way marcate con `leisure=park` e `name="Parco Mangoni"`

- ! Se conosco l'ID OSM dell'oggetto posso sempre referenziarlo direttamente con `node`, `way` e `rel` seguiti dall'ID fra parentesi. Es. `node(10020)`, `way(9865632)`

# Statements: nwr

Lo statement nwr consente di ottenere nodi, way e relazioni con una sola istruzione:

```
(node [leisure=park] ;  
way [leisure=park] ;  
rel [leisure=park] ;) ;
```



```
nwr [leisure=park] ;
```

# Statements: area

Gli oggetti **area**:

- ▶ tipo di dato che esiste solo in Overpass API
- ▶ sono un modo pratico per identificare i poligoni chiusi
- ▶ in overpass molti poligoni chiusi sono aree, ma non tutti (per le regole vedere sezione link)
- ▶ non hanno dati geografici, ma solo tags

**!** Il bbox è sempre il globo terrestre (inutile indicarlo)... Quindi occhio ai filtri!

```
//bbox=pieno Oceano Indiano
[out:xml] [bbox:-25.3,69.8,-14.4,86.8];
area[name=Torino] ["admin_level"]=8;
out;
```



```
<area id="3600043992">
  <tag k="admin_level" v="8"/>
  <tag k="boundary"
v="administrative"/>
  <tag k="name" v="Torino"/>
  <tag k="name:ca" v="Torí"/>
  <tag k="name:en" v="Turin"/>
  ....
  <tag k="type" v="boundary"/>
  <tag k="wikidata" v="Q495"/>
  <tag k="wikipedia"
v="it:Torino"/>
</area>
```



# Statements: filtri base 1/2

Altri operatori per i filtri:

Filtro	Significato
<code>node [name!=Municipio]</code>	Tutti i nodi il cui nome NON è Municipio
<code>way [oneway]</code>	Tutte le way che hanno il tag oneway (a prescindere dal suo valore)
<code>way [!oneway]</code>	Tutte le way che NON hanno il tag oneway

# Statements: filtri base 2/2

## Espressioni regolari:

Filtro	Significato
<code>way[name~"XXV aprile"]</code>	Tutte le way che contengono XXV aprile
<code>way[name~"XXv ApRILe",i]</code>	Tutte le way che contengono XXV aprile <u>a prescindere da maiuscole o minuscole</u>
<code>rel[name!~"EuroVelo"]</code>	Tutte le relazioni che non hanno nel nome la parola EuroVelo
<code>node[amenity=place_of_worship][name~"s\\. ",i]</code>	Tutti i luoghi di preghiera con il nome che contiene l'abbreviazione s. (essendo . un carattere speciale delle regex è necessario usare la sequenza di escape \\)
<code>node[~"^name (:en :es) \$"~". "]</code>	Tutti i nodi che hanno un nome in inglese o spagnolo

# Statements: definire un bbox

Come seleziono l'area su cui lavorano gli statements ?

1. Definisco un bounding box a livello di statement specificandone le coordinate

```
way["leisure"="park"] (45.35,7.977,45.45,8.144);
```

oppure utilizzando l'area selezionata visivamente in JOSM o Overpass Turbo

```
way["leisure"="park"] ({{bbox}})
```

2. Definisco un bounding box nei settings

```
[out:xml] [timeout:60] [bbox:45.35,7.977,45.45,8.144];
```

3. Definisco un bounding box per nome

A 

```
{{geocodeArea:Trino}}->.searchArea;  
way["leisure"="park"] (area.searchArea);
```

B 

```
area[name=Trino]->.searchArea;  
way["leisure"="park"] (area.searchArea);
```

C 



```
area[name=Trino];  
way(area) ["leisure"="park"];
```

Il nome viene cercato tramite Nominatim  
(viene considerata la prima corrispondenza)

Il nome viene cercato fra i "metaoggetti" area (vengono considerate tutte le corrispondenze trovate)

# Statements: mostrare i dati

Per mostrare i dati si usa lo statement **out**.

Statement	Ritorna	Dati
<b>out count;</b>	Il numero di oggetti	<pre>{ "type": "count", "id": 0,   "tags": {     "nodes": "0", "ways": "35", "relations": "0", "total": "35"   }} </pre>
<b>out center;</b>	Centroide degli oggetti	 <pre>{ "type": "way", "id": 363610976, "center": {"lat": 45.121541,"lon":7.816612} ... "tags": {   "leisure": "park" } </pre>
<b>out skel;</b>	Le struttura geometrica con le coordinate degli oggetti	 <pre>{ "type": "way", "id": 363610976, "nodes": [ 3677803034,... ]} </pre>
<b>out body;</b> (equivale a <b>out;</b> )	Le struttura geometrica con le coordinate <u>e i tag</u>	 <pre>{ "type": "way", "id": 363610976, "nodes": [ 3677803034,... ],   "tags": { "leisure": "park" } } </pre>
<b>out meta;</b>	Le struttura geometrica con le coordinate, i tag <u>e i metadati</u>	 <pre>{ "type": "way", "id": 363610976, "id": 363610976, "timestamp": "2017-12-06T15:06:40Z", "version": 3, "changeset": 12345678, "user": "xyzyz",   "uid": 12345, "nodes": [ 3677803034,... ],   "tags": {     "leisure": "park"   } } </pre>

D  
e  
t  
t  
a  
g  
g  
i  
o

NB aggiungendo 'qt' in fondo allo statement (es. **out meta qt;**) è possibile velocizzare le query. Per approfondimenti <https://wiki.openstreetmap.org/wiki/QuadTiles>

# Statements: esempi

Qualche esempio per fissare i concetti:

Esempio	Link
Farmacie a Palermo (uso di area, geocodearea e filtri di esistenza)	<a href="https://overpass-turbo.eu/s/Qzf">https://overpass-turbo.eu/s/Qzf</a>
Parchi a Ravenna (uso di bbox, nwr e differenti tipi di out)	<a href="https://overpass-turbo.eu/s/Qzg">https://overpass-turbo.eu/s/Qzg</a>
Via XXV Aprile a prescindere da come è scritta (uso di regexp)	<a href="https://overpass-turbo.eu/s/Qzh">https://overpass-turbo.eu/s/Qzh</a>
Nodi con name:en o name:es (uso di regexp, velocità rispetto a filtro classico)	<a href="https://overpass-turbo.eu/s/Qzi">https://overpass-turbo.eu/s/Qzi</a>
Nomi con parole che iniziano con minuscole (uso di regexp)	<a href="https://overpass-turbo.eu/s/QFm">https://overpass-turbo.eu/s/QFm</a>
Tutti i luoghi di preghiera con il nome che include l'abbreviazione S. (uso di regexp)	<a href="https://overpass-turbo.eu/s/QFo">https://overpass-turbo.eu/s/QFo</a>
CAP non corretti (uso di regexp)	<a href="https://overpass-turbo.eu/s/QFv">https://overpass-turbo.eu/s/QFv</a>

# Statements: recurse 1/4

## Perchè parliamo di recurse ?

Perchè gli statement di query **ritornano esclusivamente ciò che gli viene richiesto:**

- ▶ **way** ritorna delle way, ma non i riferimenti geografici ai nodi che la compongono
- ▶ **node** ritorna dei nodi, ma non le eventuali way che passano per quei nodi
- ▶ **rel** ritorna relazioni, ma non i dettagli geografici dei loro membri

# Statements: recurse 2/4

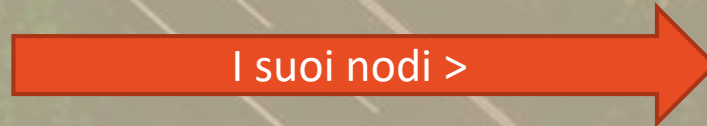
Recurse up o down ?



way

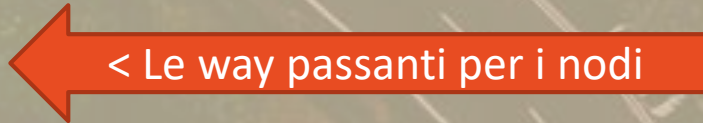


I suoi nodi >

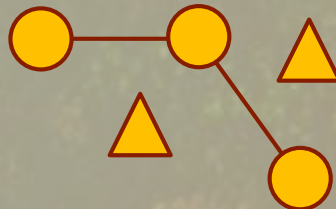


nodi

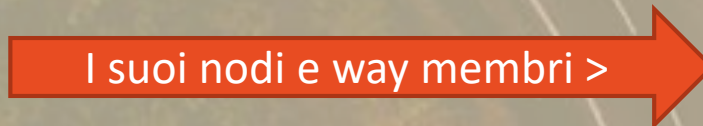
< Le way passanti per i nodi



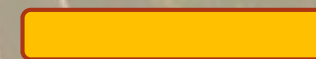
relazioni



I suoi nodi e way membri >

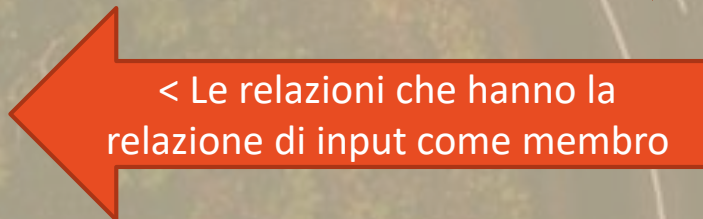


nodi




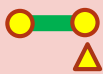


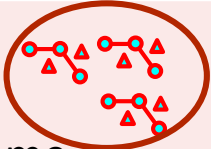

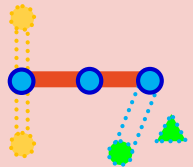



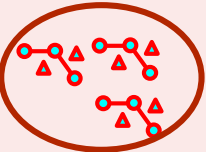
way

< Le relazioni che hanno la relazione di input come membro



relazioni

# Statements: recurse 3/4

Statement	Input	Output
> (recurse down)	<b>way</b>  <b>rel</b> 	Nodi della way  Nodi e way membri della relazione 
>> (recurse down relazioni)	<b>rel</b> con relazioni come membri 	Nodi e way membri di tutte le relazioni 
< (recurse up)	<b>node</b> <b>way</b> 	Tutte le way passanti per uno dei nodi  Tutte le relazioni che hanno come membro un nodo o una way 
<< (recurse up relazioni)	<b>rel</b> 	Le relazioni che contengono come membri una delle relazioni in input 



# Statements: recurse 4/4

Statement	Input	Output	Esempio
<code>node (w) ;</code>	way	nodi delle way	
<code>node (r) ;</code> <code>node (r: "ruolo") ;</code>	rel	nodi membri di relazioni	Tutti i nodi "fermata" di una linea bus (specificare ruolo per filtrare ulteriormente. Es. <code>node (r: "platform") ;(*)</code> )
<code>way (r) ;</code>	rel	way membri di relazioni	Il solo percorso di una linea bus
<code>rel (r) ;</code>	rel	relazioni membri di relazioni	Tutte le linee bus di una relazione route master
<code>way (bn) ;</code>	node	way che contengono i nodi	Quali altre way intersecano i nodi di una way
<code>rel (bn) ;</code>	node	Relazioni che hanno come membri i nodi	Quali linee bus usano una certa fermata
<code>rel (bw) ;</code>	way	Relazioni che hanno come membri le way	Quali linee bus passano da una certa strada
<code>rel (br) ;</code>	rel	Relazioni che hanno come membri le relazioni	La relazione route master che contiene la relazione di una linea bus

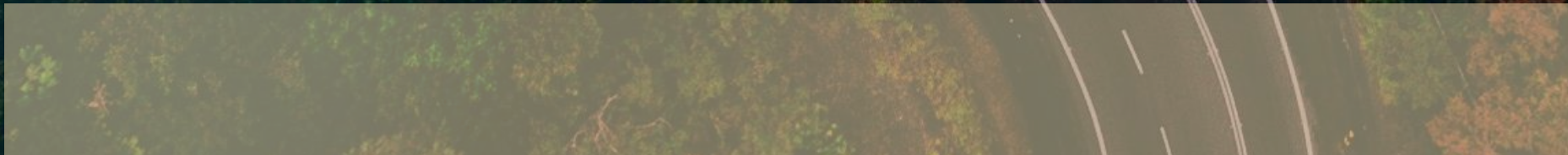
(\*) filtro per ruolo applicabile anche a `way (r)` e a tutti gli statement `rel`

# Statements: esempi ricorsione

Qualche esempio per fissare i concetti:

Esempio	Link
Nodi (effetto di > e <)	<a href="https://overpass-turbo.eu/s/Q4Y">https://overpass-turbo.eu/s/Q4Y</a>
Way (effetto di > e <)	<a href="https://overpass-turbo.eu/s/Q4X">https://overpass-turbo.eu/s/Q4X</a>
Relazioni (effetto di >, <, >> e <<)	<a href="https://overpass-turbo.eu/s/Q4S">https://overpass-turbo.eu/s/Q4S</a>
Tutti i nodi "fermata" di una linea bus (uso di <b>node (r:platform)</b> )	<a href="https://overpass-turbo.eu/s/QBr">https://overpass-turbo.eu/s/QBr</a>
Il solo percorso di una linea bus (uso di <b>way (r)</b> )	
Quali altre way intersecano i nodi di una way (uso di <b>way (bn)</b> )	<a href="https://overpass-turbo.eu/s/QBt">https://overpass-turbo.eu/s/QBt</a>
Quali linee bus usano una certa fermata (uso di <b>rel (bn)</b> )	<a href="https://overpass-turbo.eu/s/QBu">https://overpass-turbo.eu/s/QBu</a>
Quali linee bus passano da una certa strada (uso di <b>rel (bw)</b> )	<a href="https://overpass-turbo.eu/s/QBv">https://overpass-turbo.eu/s/QBv</a>
La relazione route master che contiene la relazione di una linea bus (uso di <b>rel (br)</b> )	<a href="https://overpass-turbo.eu/s/QBw">https://overpass-turbo.eu/s/QBw</a>

# Sets



# Sets

Il set è definito come . seguito da un nome arbitrario (case sensitive!).  
Per memorizzare le informazioni in un set specifico si usa l'operatore ->

```
way["leisure"="park"]->.IMieiParchi;
```

Usare le informazioni presenti in un set specifico:

- ▶ selezionare le way presenti nel set **.IMieiParchi** e filtrare solo quelle che hanno un tag name

```
way.IMieiParchi["name"];
```

- ▶ usare il set come output

```
1) .IMieiParchi out;
```

```
2) (.IMieiParchi;>);  
   out;
```

Es: <https://overpass-turbo.eu/s/QPG>

# Sets

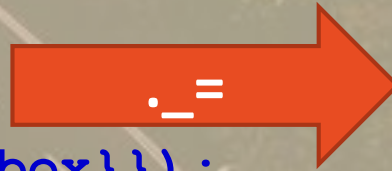
Il default set `._` è usato implicitamente quando non ne viene indicato uno

`way["leisure"="park"]`; equivale a `way["leisure"="park"]->._`;

Il set `._` viene sempre sovrascritto:

`way["leisure"="park"] ({{bbox}})`;

`way["leisure"="park"] ["name"] ({{bbox}})`;



An aerial photograph of a winding asphalt road through a dense forest. The road curves from the top left towards the bottom right. Several cars are visible on the road: a white car in the upper left, a white car in the middle, a blue car in the lower right, and another white car at the bottom. The forest is lush and green, with some trees showing autumnal colors. A semi-transparent horizontal bar is overlaid across the middle of the image.

# Block statements

# Block statements: union

Come faccio ad avere un unico insieme di dati eterogenei da trattare ? Uso la union, ovvero metto gli statement che mi interessano fra parentesi.

```
(  
    way[leisure=park] ;  
    node[amenity=bar] ;  
    rel[route=hiking] ;  
);
```

Output=way+nodi+relazioni con i tag indicati. Posso anche memorizzare i dati in un set:

```
(way[leisure=park] ; node[amenity=bar] ; rel[route=hiking] ;) -> .IMieiParchi ;
```

Esempio "notevole":

```
(. _ ; > ; ) ;
```

# Block statements: difference

Il block statement difference serve a trovare gli oggetti presenti in un insieme A che non appartengono ad un insieme B.



La sintassi è: `(A; - B;)`

Esempio: tutte le way secondary al di fuori di un certo bbox

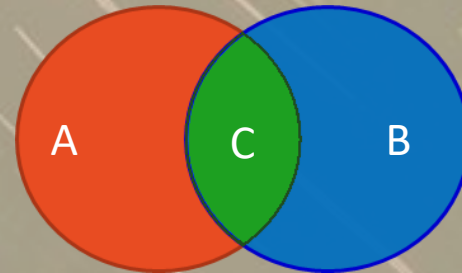
```
(way[highway=secondary]; - way(45.209,10.717,45.306,10.887););
```

Es: <https://overpass-turbo.eu/s/QbQ>



# Intersezione

L'insieme intersezione C fra A e B consiste dei soli oggetti che appartengono ad entrambi gli insiemi:



Non si ottiene con un block statement, ma con un singolo statement la cui sintassi è:

```
node . a . b ;
```

```
way . a . b ;
```

Esempi:

Nodi <https://overpass-turbo.eu/s/Qzm>, <https://overpass-turbo.eu/s/QBh>

Way <https://overpass-turbo.eu/s/Qzn>

# Block statements: foreach

Foreach consente di "trattare" l'input set un elemento alla volta invece che come un insieme unico. Supponiamo di avere la seguente query che ritorna 3 way:

```
way[highway=secondary] ;  
(. _ ;> ;) ;  
out;
```

`._=way1, way2, way3`

Ora applichiamo allo stesso statement un foreach

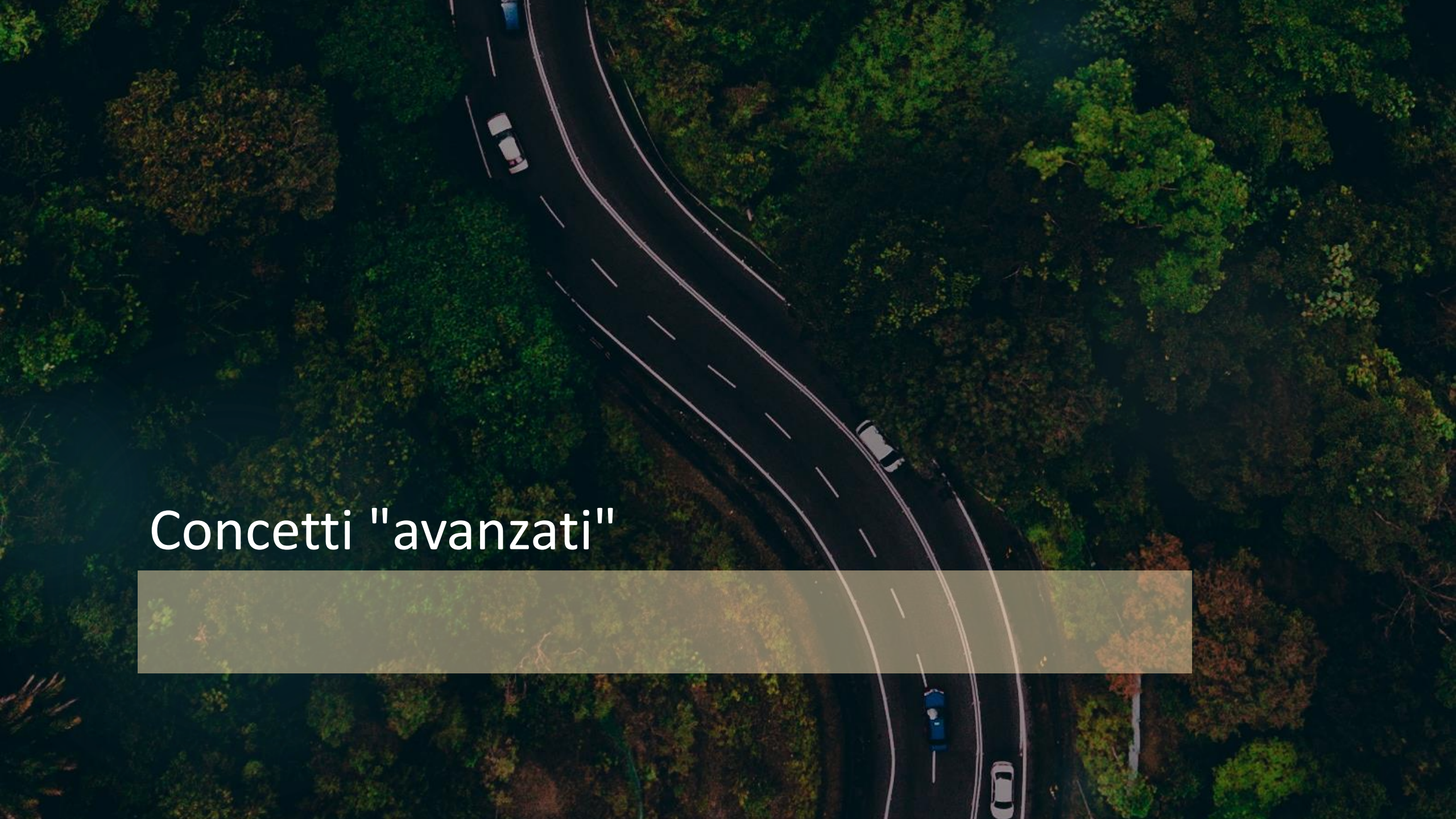
```
way[highway=secondary] ;  
foreach {  
  (. _ ;> ;) ;  
  out;  
}
```

`._=way1 al primo passaggio, way2 al secondo, ...`

Normalmente serve per la creazione di tabelle nel caso di output di tipo CSV (esempi nelle prossime slide)

- ! Se devo trattare il set `.MioSet` invece del default uso `foreach.MioSet`
- Se voglio referenziare con il nome `.MioElemento` il singolo elemento uso `foreach ->.MioElemento`

<https://overpass-turbo.eu/s/QBi>

An aerial photograph of a winding asphalt road cutting through a dense, lush green forest. The road curves from the top left towards the bottom right. Several cars are visible on the road: a white car in the upper left, a white car in the middle, a blue car in the lower right, and another white car at the very bottom. The forest is thick with trees, showing various shades of green. A semi-transparent horizontal bar is overlaid across the middle of the image, containing the text.

Concetti "avanzati"

# Block statements: "gli altri"

"Da qualche parte, qualcosa di incredibile attende di essere conosciuto" - Carl Sagan, scienziato e divulgatore

Block statement	Spiegazione
<pre>for (&lt;Evaluator&gt;) { &lt;statements&gt;}</pre>	<p>Il set in input a for viene suddiviso in sottoinsiemi omogenei in funzione di &lt;Evaluator&gt;. Ad ogni iterazione gli statements vengono eseguiti sul sottoinsieme</p> <p><a href="https://overpass-turbo.eu/s/QJC">https://overpass-turbo.eu/s/QJC</a></p>
<pre>if (&lt;Evaluator&gt;) { &lt;statements&gt; } else { &lt;statements&gt; }</pre>	<p>Il set di input viene valutato in funzione di &lt;Evaluator&gt; e gli statements eseguiti in funzione dell'esito</p> <p><a href="https://overpass-turbo.eu/s/QPH">https://overpass-turbo.eu/s/QPH</a></p>
<pre>complete { &lt;statements&gt; } complete (i) { &lt;statements&gt; } }</pre>	<p>Gli statements vengono eseguiti fino a che gli elementi in input sono stati tutti "trattati", oppure fino a quando il numero di iterazioni <i>i</i> è stato raggiunto. L'output degli statements si aggiunge all'input nell'iterazione successiva</p> <p><a href="https://overpass-turbo.eu/s/QfZ">https://overpass-turbo.eu/s/QfZ</a></p>
<pre>retro (data) { &lt;statements&gt; } }</pre>	<p>Retro esegue gli statements, ma ritornando elementi allo stato in cui erano nel giorno e ora indicati da <i>data</i> (può essere usato il valore di un campo oppure una data UTC in formato ISO es: 2020-02-22T11:30:49Z)</p> <p><a href="https://overpass-turbo.eu/s/Qg6">https://overpass-turbo.eu/s/Qg6</a></p>
<pre>compare (delta: t [tag] )</pre>	<p>Permette di valutare come uno specifico tag degli elementi in input è cambiato fra due date.</p> <p><a href="https://overpass-turbo.eu/s/Qi6">https://overpass-turbo.eu/s/Qi6</a></p>

# Filtri speciali: around

**around** filtra tutti gli oggetti ad una certa distanza da quelli del set di input.

Es: tutti i parcheggi ad un massimo di 100 metri dalle fermate dell'autobus a Belluno:

```
area [name="Belluno"] ["admin_level"=8];  
node (area) [highway=bus_stop];  
nwr [amenity=parking] (around:100);  
(. _;>);  
out meta;
```

<https://overpass-turbo.eu/s/Qj3>

E tutti i parcheggi che stanno a meno di x metri da un percorso arbitrario ?

Posso usare una linestring composta da coppie lat/lon (oppure potrei estrarre tutti i nodi delle way che mi interessa percorrere e usare la sintassi riportata sopra...)

```
nwr ["amenity"="parking"] (around:300, lat1, lon1, lat2, lon2, lat3, lon3, ...);
```

<https://overpass-turbo.eu/s/Qj2>

# Filtri speciali: user

**user** filtra tutti gli oggetti che sono stati modificati l'ultima volta da uno o più utenti specificati.

Es. Tutti i waterway modificati dagli utenti LosPollosHermanos e WWhite:

```
way["waterway"] (user: "LosPollosHermanos", "WWhite") ;  
(._i>);  
out meta;
```

# Filtri speciali: if 1/2

Il filtro **if**: consente la selezione di oggetti che soddisfano alcune condizioni determinate da un evaluator (ovvero da una funzione).

Il filtro è applicabile ad un query statement node, way, rel, nwr e area. Sintassi di esempio:

```
node(if: <evaluator>)
```

Posso usare operatori booleani:

```
way(if: <evaluator>||<evaluator>) per un OR
```

```
rel(if: <evaluator>&&<evaluator>) per un AND
```

```
node(if: !<evaluator>) per un NOT
```

Posso usare if: anche assieme a filtri tradizionali:

```
way[highway=pedestrian](if: is_closed())(bbox);
```

# Filtri speciali: if 2/2

Evaluator	Output
<code>way[highway=pedestrian] (if: <b>is_closed()</b>)</code>	Way di tipo pedestrian, ma solo se poligoni chiusi <a href="https://overpass-turbo.eu/s/QtU">https://overpass-turbo.eu/s/QtU</a>
<code>way[maxspeed] (if: <b>!is_number(t["maxspeed"])</b>)</code>	Way in cui il valore maxspeed non è un numero. Il filtro [maxspeed] seleziona le way che hanno il tag maxspeed. <code>t[nometag]</code> ritorna il valore del tag <i>nometag</i> . <a href="https://overpass-turbo.eu/s/QtQ">https://overpass-turbo.eu/s/QtQ</a>
<code>rel[route=hiking] (if: <b>length()</b>&lt;500)</code>	Relazioni che rappresentano percorsi escursionistici, ma solo se hanno una lunghezza inferiore a 500 m. Nel caso di relazioni <code>length()</code> ritorna la somma di tutte le lunghezze dei membri di tipo way <a href="https://overpass-turbo.eu/s/QtR">https://overpass-turbo.eu/s/QtR</a>
<code>node (if: <b>version()</b>==1&amp;&amp;<b>user()</b>=="IlBano")</code>	Nodi create dall'utente IlBano e rimasti immutati <a href="https://overpass-turbo.eu/s/QtS">https://overpass-turbo.eu/s/QtS</a>
<code>way (if: <b>count_tags()</b> == 0)</code>	Le way senza tag (attenzione ai falsi positivi=way membri di relazioni) <a href="https://overpass-turbo.eu/s/QtT">https://overpass-turbo.eu/s/QtT</a>
<code>way (if: <b>count_members()</b>&gt;1900)</code>	Way con più di 1900 nodi. (se applicato ad una relazione restituisce il numero di membri) <a href="https://overpass-turbo.eu/s/QtV">https://overpass-turbo.eu/s/QtV</a>
<code>rel["type"="multipolygon"] (if: <b>count_by_role("outer")</b> &lt; 1);</code>	Relazioni multipolygon senza membri outer (errore!) <a href="https://overpass-turbo.eu/s/QtW">https://overpass-turbo.eu/s/QtW</a>



# Filtri speciali: pivot

L'uso del filtro **pivot** è strettamente legato agli oggetti **area** che sappiamo essere privi di riferimenti geografici (no lat/lon di way o nodi). Con **pivot** li recuperiamo

```
area[name="Piazzale Napoleone I"];  
out;
```



```
<area id="2601608822">  
  <tag k="area" v="yes"/>  
  <tag k="bicycle" v="yes"/>  
  <tag k="highway" v="pedestrian"/>  
  <tag k="lit" v="yes"/>  
  <tag k="name" v="Piazzale Napoleone I"/>  
  <tag k="surface" v="pebblestone"/>  
</area>
```

```
area[name="Piazzale Napoleone I"];  
way(pivot);  
(. _;>);  
out;
```



```
<node id="267941845" lat="41.9114711" lon="12.4781341"/>  
...  
<way id="201608822">  
  <nd ref="2008994923"/>  
  ...  
  <nd ref="2116374401"/>  
  <nd ref="2008994923"/>  
  <tag k="area" v="yes"/>  
  <tag k="bicycle" v="yes"/>  
  <tag k="highway" v="pedestrian"/>  
  <tag k="lit" v="yes"/>  
  <tag k="name" v="Piazzale Napoleone I"/>  
  <tag k="surface" v="pebblestone"/>  
</way>
```

A cosa serve se potremmo usare la query `way[name="Piazzale Napoleone I"];?`  
... fra un paio di slide

# Statement map\_to\_area

Lo statement `map_to_area` permette di estrarre gli oggetti area che rappresentano way e relazioni in input (ovvero esegue logicamente l'operazione inversa di `pivot`).

Utile per selezionare aree in modo molto selettivo e sfruttando le gerarchie implicitamente presenti in OSM.

Esempio: estrarre i comuni della provincia di Teramo

```
//Seleziono la sola l'area che rappresenta la provincia
area["name"="Teramo"]["admin_level"]=6]->.ProvinciaTE;
//seleziono le relazioni di tipo comune nell'area
rel(area.ProvinciaTE)["admin_level"]=8];
//ottengo le aree che li rappresentano
map_to_area;
```

Esempi nella sezione "Dati in tabella 2/2"

# Query is\_in

**is\_in** è una query che:

- ▶ Prende i nodi (solo quelli) in input dal default set (oppure da un set specifico)
- ▶ Restituisce in output tutte le aree che contengono quei nodi (solo aree – no way, nodi o rel)

Per prendere i nodi da **.MioSet** e mettere l'output in **.MieAree**

```
.MioSet is_in ->.MieAree
```

**!** Come cerco le aree che contengono una way ? Ottengo prima i nodi della way con un recurse down >:

```
way[highway=residential][name="Via Lattea"];
```

```
>;
```

```
is_in->.AreeConViaLattea
```

**is\_in** assieme a **pivot** consente di effettuare query complesse come ad esempio:trova tutti i comuni di una regione che NON hanno un **amenity=townhall**

<https://overpass-turbo.eu/s/Qkg>

# Dati dal passato (di OSM) 1/2

Overpass consente di lavorare con dati OSM storici

▶ Filtro **newer**

```
way (newer: "2018-09-14T15:00:00Z")
```

estrae le sole way cambiate dal 14/9/2018 alle ore 15

▶ Setting **date**

```
[date: "2019-05-10T13:00:00Z"]
```

estrae i dati come apparivano al 10/5/2019 alle ore 13

▶ Setting **diff** e **adiff**

```
[diff: "2018-09-14T00:00:00Z"]
```

estrae i soli oggetti cambiati a partire dal 14/9/2018

```
[diff: "2018-09-14T00:00:00Z", "2019-09-14T23:59:59Z"]
```

estrae i soli oggetti cambiati fra il 14/9/2018 e il 14/9/2019

- ❑ L'output può essere solo xml, ma non può essere usato in JOSM
- ❑ Gli oggetti hanno un tag che identifica se sono stati aggiunti, cancellati o modificati
- ❑ **adiff** usato al posto di **diff** evidenzia dati aggiuntivi in caso di cancellazioni

# Dati dal passato (di OSM) 2/2

## newer vs diff vs adiff

```
[out:xml] [timeout:90];  
way (newer:"2018-09-  
14T15:00:00Z") ({{bbox}});  
out meta;
```

```
<way id="95365689" version="9"  
timestamp="2019-01-13T17:17:27Z"  
changeset="66279635" uid="1720269"  
user="IlBano">  
  <nd ref="766697399"/>  
  <nd ref="6055906686"/>  
  ...  
  <tag k="highway" v="tertiary"/>  
  <tag k="name" v="Strada Regionale  
82"/>  
  <tag k="ref" v="SR82"/>  
  <tag k="surface" v="asphalt"/>  
</way>  
<way id="95366674" version="10"  
timestamp="2019-12-25T14:16:08Z"  
changeset="78848459" uid="1758654"  
user="JJJWegdam">  
  <nd ref="1106033915"/>  
  <nd ref="1106033923"/>  
  ...
```

```
[diff:"2018-09-14T15:00:00Z", "2019-  
09-  
14T15:00:00Z"] [out:xml] [timeout:90];  
way ({{bbox}});  
out meta;
```

```
<action type="delete">  
<old>  
  <way id="61404117" version="11" timestamp="2014-  
02-07T05:03:17Z" changeset="20424036" uid="893327"  
user="mcheckimport">  
  <nd ref="766697399"/>  
  ...  
  <tag k="highway" v="tertiary"/>  
  ...  
</way>  
</old>  
</action>  
<action type="modify">  
<old>  
  <way id="95365689" version="5" timestamp="2018-08-  
09T17:26:44Z" changeset="61508287" uid="6683305"  
user="Tigranych">  
  <nd ref="766697413"/>  
  ...  
  <tag k="highway" v="tertiary"/>  
</way>  
</old>  
<new>  
  <way id="95365689" version="9" timestamp="2019-01-  
13T17:17:27Z" changeset="66279635" uid="1720269"  
user="IlBano">  
  <nd ref="766697399"/>
```

```
[adiff:"2018-09-14T15:00:00Z", "2019-09-  
14T15:00:00Z"] [out:xml] [timeout:90];  
way ({{bbox}});  
out meta;
```

```
<action type="delete">  
<old>  
  <way id="61404117" version="11" timestamp="2014-02-07T05:03:17Z"  
changeset="20424036" uid="893327" user="mcheckimport">  
  <nd ref="766697399"/>  
  ....  
  <tag k="highway" v="tertiary"/>  
  ...  
</way>  
</old>  
<new>  
  <way id="61404117" visible="false" version="13" timestamp="2018-  
11-12T21:26:25Z" changeset="64421953" uid="1720269" user="IlBano"/>  
</new>  
</action>  
<action type="modify">  
<old>  
  <way id="95365689" version="5" timestamp="2018-08-09T17:26:44Z"  
changeset="61508287" uid="6683305" user="Tigranych">  
  <nd ref="766697413"/>  
  ...  
  <tag k="highway" v="tertiary"/>  
</way>  
</old>  
<new>  
  <way id="95365689" version="9" timestamp="2019-01-13T17:17:27Z"  
changeset="66279635" uid="1720269" user="IlBano">  
  <nd ref="766697399"/>
```

<https://overpass-turbo.eu/s/QqY>

# Dati in tabella 1/2

Abbiamo già visto che CSV può essere un tipo di output. Viene controllato dal setting out con la seguente sintassi:

```
[out:csv(campo1, campo2, ...<,intestazione><,separatore>)]
```

Dove:

- ▶ intestazione indica se far apparire in output il nome dei campi (default=true) oppure no (false) - opzionale
- ▶ separatore è il carattere di separazione fra campi (default=tabulazione) - opzionale
- ▶ campoX è il nome di un tag oppure un valore speciale:

Valore	Descrizione
::id	id OSM dell'oggetto
::type	Tipo oggetto OSM: node, way, relation
::lat ::lon	Latitudine/longitudine dell'oggetto (solo se node o se selezionato out center)
::version	Versione dell'oggetto OSM

Valore	Descrizione
::timestamp	data e ora in cui l'oggetto è stato modificato l'ultima volta
::changeset	ID del changeset in cui l'oggetto è stato modificato l'ultima volta
::uid ::user	ID utente e nome utente OSM
::count	Numero totale di node, way, relation e area esaminati (si possono contare anche specifici oggetti con ::count:nodes, ::count:ways, ::count:relations e ::count:areas)

<https://overpass-turbo.eu/s/QiZ>

# Dati in tabella 2/2

Con gli statement **make** e **convert** è possibile creare dei campi calcolati.

- ▶ **make** genera una sola riga per set di input e si usa in genere per creare statistiche di insiemi di dati
- ▶ **convert** genera una riga per ogni elemento del set di input

La sintassi è:

```
make <tipo> campo1=<evaluator>, campo2=<evaluator>,...
```

```
convert <tipo> campo1=<evaluator>, campo2=<evaluator>,...
```

Dove:

- ▶ tipo è un valore arbitrario (spesso si vede stat, count, node o way)
- ▶ campoX è il nome del campo riportato nella definizione out:csv

Esempio di uso:

```
[out:csv (::id,name,piedi,metri)];
```

```
<query per selezionare delle way>
```

```
convert stat ::id=id(),piedi=(length()/1000*0.62*5280),metri=length(),name=t["name"];
```

Esempio: differenza fra make e convert <https://overpass-turbo.eu/s/Qyp>

Esempio: output con uso di foreach e map\_to\_area <https://overpass-turbo.eu/s/QJH>

# Links

Language reference di Overpass QL

[https://wiki.openstreetmap.org/wiki/Overpass\\_API/Overpass\\_QL](https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL)

Esempi di Overpass QL

[https://wiki.openstreetmap.org/wiki/Overpass\\_API/Overpass\\_API\\_by\\_Example](https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_API_by_Example)

Verifica raggiungimento quota su Overpass API

<http://www.overpass-api.de/api/status>

Blog di Overpass API (attualmente non più disponibile)

<https://www.overpass-api.de/blog/>

Tool per creare linestring (Attenzione! Genera coordinate lon/lat invece che lat/lon come richiesto da filtro around di Overpass)

<https://www.keene.edu/campus/maps/tool/>

Regole di creazione degli oggetti area in Overpass

[https://wiki.openstreetmap.org/wiki/Overpass\\_turbo/Polygon\\_Features](https://wiki.openstreetmap.org/wiki/Overpass_turbo/Polygon_Features)

Costruire un bbox

<http://norbertrenner.de/osm/bbox.html>



# Credits

Presentazione creata da Andrea Albani e sottoposta a licenza CC-BY-SA 4.0



Un ringraziamento a Andrea Musuruane, Marco Brancolini e Ferruccio Cantone per le considerazioni/consigli/stimoli forniti durante la stesura.

Foto di background di Deva Darshan disponibile con licenza reperibile qui <https://www.pexels.com/it-it/photo-license/>

Presentazione reperibile qui



An aerial photograph of a winding asphalt road cutting through a dense, lush green forest. Several cars are visible on the road, including a white car, a blue car, and a dark car. The road curves from the top left towards the bottom right. The text 'Grazie!' is overlaid in a large, bold, yellow font across the center of the image. Below the text is a semi-transparent horizontal bar.

Grazie!