OpenSchoolMaps: OpenStreetMap-Daten extrahieren mit SQL und Postpass

Stefan Keller und Jonas Grüter, im Juni 2026

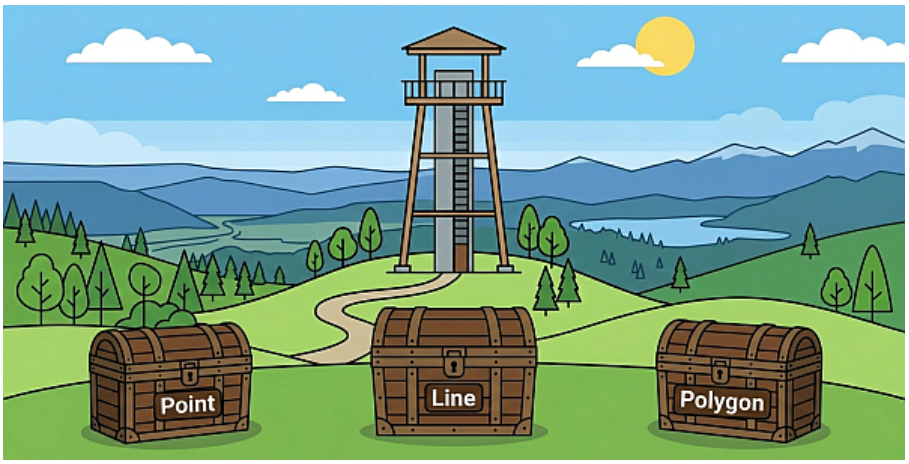

Figure 1. Created by Gemini.

OpenStreetMap (OSM) enthält detaillierte Geodaten über die ganze Welt – frei verfügbar und laufend aktualisiert. Mit **Postpass** lassen sich diese Daten direkt mit der bekannten Computersprache SQL abfragen, ohne aufwändige Installation oder Datei-Downloads.

In diesem **Tutorial** lernst du in wenigen Schritten, wie man Daten wie zum Beispiel Aussichtstürme und Restaurants der Schweiz aus einer weltweiten Datenbank filtert und extrahiert.

Du brauchst zur **Vorbereitung** nur einen Browser, ein OpenStreetMap-Konto, sowie Freude am Ausprobieren und etwas technisches Flair. Es werden keine SQL-Kenntnisse vorausgesetzt.

Noch kein OpenStreetMap-Konto? Einfach auf www.openstreetmap.org gehen und eines anlegen (OSM ist sehr datenschutzkonform). Mehr zu OpenStreetMap gibt es beim ersten Einloggen und in den OpenSchoolMaps-Tutorials.

Überblick über die einfachen ersten Schritte:

1. Verstehen, wie OSM-Daten strukturiert sind.
2. Einige datenbanktechnische Begriffe kennenlernen
3. Erste räumliche Abfragen mit Aussichtstürmen
4. OSM-Daten exportieren und testhalber visualisieren
5. Fortgeschrittene Abfrage-Beispiele (Restaurants, Waldwege).

Wer mehr hinter diese SQL-Abfragen blicken möchte, für den gibt es unten ein eigenes **Kapitel "Mehr Hintergrund gefällig?"**. Dort gibt es auch mehr zu **Postpass** und warum das in vielen Fällen die bessere Abfragesprache ist als z.B. **Overpass QL**.

Datenstruktur in OpenStreetMap: Von Objekten und Tags

Um das Extrahieren von Daten aus OpenStreetMap zu verstehen, erklären wir zuerst kurz, wie Daten in OSM strukturiert sind.

Jedes Objekt in der OpenStreetMap-Datenbank (z.B. der Aussichtsturm Bachtelturm) wird durch seine geografische Lage und eine Liste von Eigenschaften beschrieben. Diese Eigenschaften nennen wir in OSM Tags. Tags bestehen aus einem Schlüssel (Key) und einem Wert (Value). Die geografische Lage ist als Geometrie gespeichert, vom Typ Punkt, Linie oder Fläche (Englisch: Point, Line, Polygon). Die genaueren Strukturen von OSM sind im Kapitel mit mehr Hintergrund weiter unten beschrieben.

Haupt-Tags (Kategorien)

Um ein Objekt wie den Bachtelturm zu definieren, benötigt es einen Haupt-Tags, der festlegt, um was es sich handelt. Dieser Tag dient als Kategorie für die Extraktion.

- Struktur: Schlüssel=Wert
- Beispiel: Aussichtsturm Bachtelturm: `man_made=tower` und `tower:type=observation`

Ohne diesen Haupt-Tag wäre das Objekt für die meisten Anwendungen undefiniert. Wenn du Daten extrahierst, filterst du im ersten Schritt fast immer nach einem Haupt-Tag.

Sekundärtags und weitere Tags

Alle weiteren Informationen werden dem Objekt nach demselben Prinzip der Tags, d.h. Schlüssel-Werte-Paare, hinzugefügt. Es gibt keine explizite Unterscheidung zwischen Haupt- und Sekundärtags. Die Sekundärtags beschreiben einfach weitere, spezifische Merkmale des Objekts.

Der Datensatz für den Bachtelturm sieht zum Beispiel so aus:

Key (Schlüssel)	**Value (Wert)**	**Funktion**
man_made	tower	Haupt-Tag (Kategorie)
tower:type	observation	Sekundärtag: Typ des Turms
name	Bachtelturm	Attribut: Name des Objekts
operator	Swisscom	Attribut: Betreiber
height	75	Attribut: Höhe (in Metern)

Tags sind Schlüssel-Werte-Paare

Die gesamten OpenStreetMap-Daten basieren auf Tags, d.h. Schlüssel-Wert-Paaren:

- Der **Schlüssel (Key)** beschreibt die Art der Eigenschaft (z.B. name).
- Der **Wert (Value)** beschreibt die konkrete Ausprägung dieser Eigenschaft (z.B. Bachtelturm).

Bei der Extraktion kannst du diese Struktur nutzen, um gezielt zu filtern. Du suchst beispielsweise zuerst nach dem Haupt-Tag `man_made=tower` und verfeinerst deine Suche bei Bedarf durch zusätzliche Tags wie `tower:type=observation`. Denn es gibt noch viele andere Turmtypen, wie Funkturm, Wachturm etc..

Mit folgenden Online-Tools findet man die richtigen OSM-Tags: Tagfinder, OSM-Wiki, Taginfo.

Technische Begriffe

Hier einige wichtige, datenbanktechnische Begriffe:

- **SQL:** Die Basissprache, mit der du Anfragen an die Datenbank stellst. Du sagst der Datenbank damit genau, welche Informationen du aus welcher Tabelle erhalten möchtest.
- **Spatial SQL:** Eine Erweiterung (PostGIS), die herkömmliches SQL um geografische Funktionen ergänzt. Damit kannst du Objekte nicht nur nach Namen filtern, sondern auch nach ihrer Position auf der Erde.
- **bbox (Bounding Box):** Ein rechteckiger Suchrahmen auf der Karte, der durch Koordinaten definiert wird. Er begrenzt die Abfrage auf ein bestimmtes Gebiet, damit die Datenbank nur dort nach Aussichtstürmen sucht und nicht den gesamten Planeten durchsuchen muss.
- **Key-Value-Format:** In der Spalte `tags` werden alle Eigenschaften eines Aussichtsturms (wie Kategorie, Typ oder Name) als Schlüssel-Wert-Paare gespeichert. JSONB ist eine effiziente Speichermethode, die es erlaubt, diese Paare schnell zu durchsuchen. Die Tags des Bachtelturms sehen z.B. so aus: `{"man_made": "tower", "tower:type": "observation", "name": "Bachtelturm", "operator": "Swisscom", "height": "75"}`.
- **Postpass-Tabellenschema:** Mit diesem System werden alle punktförmigen Objekte in der Tabelle „postpass_point" gespeichert. Jede Zeile enthält eine eindeutige ID (`osm_id`), die geografische Position (`geom`) und das Feld `tags`, das sowohl den Haupt-Tag als auch alle weiteren Attribute enthält. Eine vollständigere Liste der Postpass-Tabellen ist im Anhang.

Mehr zu all diesen Begriffen und Themen gibt es im Kapitel **"Mehr Hintergrund gefällig?"** unten.

Dies ist die Postpass-Homepage: postpass.geofabrik.de. Die wichtigsten Postpass-Tabellen sind im Anhang zusammengestellt.

Daten Filtern

Beim Filtern prüfen wir, ob bestimmte Tags im Feld `tags` einer Postpass-Tabelle wie z.B. `postpass_point` enthalten sind.

Starte die **Overpass Turbo-Instanz** und füge mit Copy & Paste folgende SQL-Abfrage ein. Zoome anschliessend in eine Gegend, die du kennst, z. B. ins Zürcher Oberland rund um Wetzikon ZH. Dadurch wird der räumliche Ausschnitt festgelegt, aus dem wir Aussichtstürme extrahieren möchten. Klicke dann auf "Ausführen".

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT osm_id, tags->>'name' AS "name", geom
FROM postpass_point
WHERE geom && {{bbox}}
  AND tags @> '{"man_made": "tower", "tower:type": "observation"}'
```

Listing: Postpass-Abfrage für Aussichtstürme im aktuellen Kartenausschnitt.

In der gegeben Abfrage filtern wir in der Postpass-Datenbank nach Aussichtstürmen, also den Tags bzw. den zwei Key-Value-Paaren `man_made=tower` und `tower:type=observation`. Dazu müssen wir das Spezialrechenzeichen `@>` verwenden (in SQL der sogenannte "Enthaltensein-Operator", der mit A @ B testet, ob B in A enthalten ist).

Erläuterungen:

- Aussichtstürme haben oft einen Namen. Diesen wollen wir mit dem Spezialrechenzeichen `->>` ausgeben und setzen den Feldnamen "name".
- Das `WHERE geom && {{bbox}}` legt zuerst den räumlichen Ausschnitt fest.
- Das `AND tags @> ⋯` filtert nach Aussichtstürmen. In dieser Zeichenkette `{"man_made": "tower", "tower:type": "observation"}` sind die zwei erwähnten Key-Value-Paare `man_made=tower` und `tower:type=observation` enthalten.

Mehr zur räumlichen Filterung findest du unten im Abschnitt "**Spatial SQL**"".

Erweiterte Ausgabe

Wir erweitern den Filter auf zwei Turmtypen: Aussichts- und Wachtürme.

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT osm_id, tags->>'name' AS "name", geom
FROM postpass_point
WHERE geom && {{bbox}}
  AND (
    tags @> '{"man_made": "tower", "tower:type": "observation"}'
    OR  tags @> '{"man_made": "tower", "tower:type": "defensive"}'
  )
```

Listing: Postpass-Abfrage für Beobachtungs- und Wehrtürme im aktuellen Kartenausschnitt.

Hier wurde mit dem “OR” eine logische Verknüpfung gemacht und zwar sinngemäss "entweder Aussichtsturm oder Wachturm".

Folgende Abfrage mit “IN” ist eleganter als vorherige Abfrage mit “OR”:

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT osm_id, tags->>'name' AS "name", geom
FROM postpass_point
WHERE geom && {{bbox}}
  AND tags @> '{"man_made": "tower"}'
  AND tags->>'tower:type' IN ('observation', 'defensive')
```

Listing: Postpass-Abfrage für Beobachtungs- und Wehrtürme mittels IN-Operator.

Mit Aussichtstürmen haben wir bewusst ein Punktobjekt gewählt. Postpass umfasst natürlich auch Tabellen für Linien- und Flächenobjekte.

Ist dir aufgefallen, dass hier das Spezialzeichen `->>` auch in der `WHERE`-Klausel erscheint? Mehr dazu im Kapitel "**SQL-Abfragen der Tags**"

OSM-Daten exportieren und visualisieren

Postpass gibt Abfrageergebnisse standardmässig als GeoJSON zurück – dem gängigen offenen Format für Geodaten. Du kannst das Ergebnis direkt herunterladen und anschliessend im Browser prüfen.

- Exportieren: Klicke in der Postpass-Oberfläche auf den Download-Button und speichere die Datei als `.geojson`.
- Visualisieren mit geojson.io:
  1. Öffne geojson.io im Browser.
  2. Ziehe deine `.geojson`-Datei per Drag & Drop in die Karte – oder füge den Inhalt direkt in den Editor auf der rechten Seite ein.
  3. Die extrahierten Aussichtstürme erscheinen sofort als Punkte auf der Karte.

GeoJSON ist ein offenes Format zur Darstellung geografischer Daten, das dem Standardformat JSON folgt. GeoJSON ist ein Dateiformat, das dazu dient, geografische Objekte und deren Eigenschaften in einer klar strukturierten JSON-Datei zu beschreiben. Der Einsatzbereich von GeoJSON umfasst die Erstellung von Webkarten, die Geodatenanalyse, die Bereitstellung von APIs für Standorte sowie den Austausch zwischen GIS-Tools.

Postpass gibt standartmässig immer GeoJSON zurück, falls eine CSV Datei gewünscht ist, kann man das via geojson.io machen. Die SQL-Query kann in Overpass-turbo durchgeführt werden, als JSON kopieren und in geojson.io öffnen. Dort gibt es die Möglichkeit, die Daten als CSV zu sichern.

Einen Fehler in den OSM-Daten entdeckt? Einfach Rechtsklick und eine Notiz (einen Hinweis/Kartenfehler hier melden) hinterlassen. Um selber OSM zu editieren siehe OpenSchoolMaps.

Das Restaurant-Beispiel

Fortgeschrittene Abfrage am Beispiel von Restaurants.

OSM-Objekte, die Punkte oder Flächen sein können

OSM enthält viele "Punkte von Interesse" (englisch: Points-of-Interest, POIs) unter anderem auch Restaurants und andere Lokale zum Essen & Trinken.

Aussichtstürme sind punktförmige Objekte. Restaurants hingegen können sowohl als Punkte als auch als (Gebäude-)Flächen erfasst sein. Wenn in der Abfrage oben einfach die Tags für Aussichtstürme mit dem Haupttag für Restaurants ersetzt werden, verfehlt man einige Daten in OSM.

Wegen Punkten und Flächen benötigen wir die Tabelle `postpass_pointpolygon` und müssen bei der Ausgabe mehrere Dinge beachten. Damit Restaurants, die als Fläche/Polygon erfasst sind, als Punkt ausgegeben werden, wird die Funktion `ST_PointOnSurface` (diese Funktion nimmt einen Punkt der garantiert auf der Fläche liegt) genutzt. Um erkennen zu können um welchen Dateityp es sich handelt, wird `osm_type` in `SELECT` eingefügt, damit der ursprüngliche Typ angezeigt wird und mit `osm_id` wird das Objekt identifiziert.

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT
    osm_id,
    osm_type,
    tags->>'name' AS "name",
    ST_PointOnSurface(geom) AS geom
FROM postpass_pointpolygon
WHERE geom && {{bbox}}
  AND tags @> '{"amenity": "restaurant"}'
```

Listing: Postpass-Abfrage für Restaurants im aktuellen Kartenausschnitt.

> **HINWEIS**: `ST_PointOnSurface()` ist eine räumliche SQL-Funktion, die gegeben eine FLäche einen darin enthaltenen Punkt berechnet. Wenn du mehr über solche Funktionen wissen willst, siehe unten "**APPENDIX: PostGIS-Funktionen**".

OSM-Objekte können identifiziert werden. Dies geschieht vor allem mit der `osm_id` und und dem `osm_type`:

- `osm_id`: Jedes Objekt hat einen Identifikator (ID) zugeteilt, wodurch gezielt auf Objekte zugegriffen werden kann. Man benötigt die ID zusätzlich für Verknüpfungen von Daten oder um etwas im OSM-System wiederzufinden.
- `osm_type`: Gibt an, welcher Objekttyp aus OSM bzw. welches OSM-Element vorliegt (Node (Punkt), Way (Linie, manchmal Fläche), Relation (Fläche)). Beispiel: Der Bachtelturm ist vom Typ "Punkt"

(`osm_type`) mit der Identifikation (`osm_id`) 1686470326, also https://www.openstreetmap.org/node/1686470326.

Küche: Tags mit mehreren OSM-Werten

Bei Restaurants oder Cafés interessiert die Art der angebotenen Speisen. In OSM gibt es für den "Küchen-Stil" den Tag "cuisine". Viele Restaurants bieten mehr als einen Küchen-Stil an, also z.B. Pizza und italienisch allgemein. Das führt zu einer Liste von mehreren Werten in einer einzigen Spalte, also z.B. `cuisine=pizza;italian`.

Uns interessiert der Einfachheit halber nur italienische Restaurants. Nachfolgend die Query von oben, zuerst aber die Erläuterungen dazu:

- `tags->>'cuisine'` holt den Wert aus den OSM-Tags. → italian;pizza;pasta
- `string_to_array(tags->>'cuisine', ';')` wandelt den Text in eine Liste (Array) um. → {italien, pizza, pasta}
- `@> '{italian}`: Enthält die Liste (das Array) den Wert `italian`?

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT
    osm_id,
    osm_type,
    tags->>'name'    AS "name",
    tags->>'amenity' AS amenity,
    tags->>'cuisine' AS cuisine,
    ST_PointOnSurface(geom) AS geom
FROM postpass_pointpolygon
WHERE geom && {{bbox}}
  AND tags @> '{"amenity": "restaurant"}'
  AND string_to_array(tags->>'cuisine', ';') @> '{italian}'
```

Listing: Postpass-Abfrage für italienische Restaurants im aktuellen Kartenausschnitt.

Restaurants ohne Namen (NULL-Werte)

Mit der obigen Abfrage kommt es oft vor, dass bei Restaurants kein Namen oder keine 'cuisine' erfasst wurde. Für fehlende Angaben gibt es in Datenbanken einen eigenen Wert - den sogenannten NULL-Wert.

Erläuterung:

- `AND tags->>'name' IS NOT NULL` Setzt die Bedingung, dass 'name' nicht leer sein darf.
- `COALESCE(tags->>'name', '(Name?)') AS "name"` COALESCE nimmt jeweils den ersten Wert der nicht Null ist. Nimm den Namen, wenn dieser Null ist ersetzt ihn mit name.

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT
    osm_id,
    osm_type,
    COALESCE(tags->>'name', '(Name?)') AS "name",
    tags->>'amenity'  AS amenity,
    tags->>'cuisine'  AS cuisine,
    ST_PointOnSurface(geom) AS geom
FROM postpass_pointpolygon
WHERE geom && {{bbox}}
  AND tags @> '{"amenity": "restaurant"}'
  AND string_to_array(tags->>'cuisine', ';') @> '{italian}'
  AND tags->>'name' IS NOT NULL
```

Listing: Postpass-Abfrage für italienische Restaurants mit Ersatzwert für fehlende Namen mittels COALESCE().

Alle italienischen Essensmöglichkeiten mit Adressen

Hier noch ein weiteres Beispiel mit allen Essensmöglichkeiten - also nicht nur Restaurants. Damit erweitern wir das Filter-Kriterium auf alle möglichen weiteren Kategorien neben Restaurant, wie: Café, Bar, Fast Food-Stände.

Zudem wäre es noch interessant, die Adressen zu haben. Doch Adressen sind ein komplexes Thema, viele Adress-Angaben an OSM-Objekten fehlen. Dabei ist die Adresse eigentlich in OSM erfasst - halt nur als Tag am umfassenden Gebäude. Wer wirklich alle Adressen von Restaurants erhalten will, muss sich mit Geocoding beschäftigen.

Hier eine 'ultimative' Abfrage aller italienischen Essensmöglichkeiten mit Adressen.

Erläuterung:

- `CONCAT_WS()` verbindet alle Teile mit einem Leerzeichen dazwischen

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT
    osm_id,
    osm_type,
    COALESCE(tags->>'name', '(Name?)') AS "name",
    tags->>'amenity' AS amenity,
    tags->>'cuisine' AS cuisine,
    CONCAT_WS(' ',
        COALESCE(tags->>'addr:street', tags->>'addr:place'),
        tags->>'addr:housenumber',
        tags->>'addr:postcode',
        tags->>'addr:city'
```

```
    ) AS full_address,
    ST_PointOnSurface(geom) AS geom
FROM postpass_pointpolygon
WHERE geom && {{bbox}}
  AND tags->>'amenity' IN ('restaurant', 'cafe', 'bar', 'fast_food')
  AND string_to_array(tags->>'cuisine', ';') @> '{italian}'
```

Listing: Postpass-Abfrage für italienische Gastronomiebetriebe mit zusammengesetzter Adresse mittels CONCAT_WS().

Falls du gerne besser verstehen willst, wie man genau SQL-Abfragen liest, versteht und selber erstellt, siehe unten Mehr Hintergrund gefällig?.

Weitere Beispiele

Nachfolgend zwei weitere Beispiele, eines das zwei Objektklassen in eine räumliche Beziehung setzt ("Restaurants in der Nähe...") und eines mit Strassen und Wegen (Waldwege).

Restaurants in der Nähe von Burgen der Schweiz

Genauer: Alle Restaurants in der Nähe von 50m aller Burgen der Schweiz und 500m angrenzendes Ausland.

Erläuterungen:

- `WITH`-Statement: Siehe das Kapitel "Mehr Hintergrund gefällig?" zu SQL allgemein unten sowie das entsprechende Kapitel zu WITH-Anfragen Kapitel: CTEs.
- `ST_SimplifyPreserveTopology` vereinfacht die (Schweizer) Grenze, um unnötig komplexe Geometrien zu eliminieren.
- `ST_Buffer` buffert die (Schweizer) Grenze, so dass auch grenznahe Burgen und Restaurants gefunden werden.
- Das `FROM postpass_polygon WHERE osm_type = 'R' AND osm_id = 51701` ist das OSM-Objekt der Schweiz, d.h. die Relation mit der osm_id 51701. Das findet man auf openstreetmap.org.
- `ST_Distance` und `ST_DWithin` verwenden den Trick mit dem GEOGRAPHY-Typ, so dass der Distanz-Parameter in Metern angegeben werden kann.
- `r.geom && ST_Expand(c.geom, 0.005)` filtern mit Index vor.

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

WITH swiss_border AS MATERIALIZED (
    SELECT ST_Buffer(
               ST_SimplifyPreserveTopology(geom, 0.0001)::geography, 500
           )::geometry AS geom
    FROM postpass_polygon
    WHERE osm_type = 'R' AND osm_id = 51701
```

```
),
castles AS (
    SELECT
        COALESCE(p.tags->>'name', '(?)') AS name,
        p.geom
    FROM postpass_pointpolygon p, swiss_border s
    WHERE p.tags @> '{"historic": "castle"}'
      AND ST_Within(p.geom, s.geom)
),
restaurants AS (
    SELECT
        p.osm_id,
        p.osm_type,
        COALESCE(p.tags->>'name', '(?)') AS name,
        COALESCE(p.tags->>'cuisine', '(?)') AS cuisine,
        COALESCE(p.tags->>'website', '(?)') AS website,
        p.geom
    FROM postpass_pointpolygon p, swiss_border s
    WHERE p.tags @> '{"amenity": "restaurant"}'
      AND ST_Within(p.geom, s.geom)
)
SELECT DISTINCT ON (r.osm_id)
    r.osm_id,
    r.osm_type,
    r.name,
    r.cuisine,
    r.website,
    ST_Distance(r.geom::geography, c.geom::geography)::int AS distance_meter,
    c.name AS near,
    ST_PointOnSurface(r.geom) AS geom
FROM restaurants AS r
JOIN castles AS c
  ON r.geom && ST_Expand(c.geom, 0.005)
 AND ST_DWithin(r.geom::geography, c.geom::geography, 50)
```

Listing: Komplexe Postpass-Abfrage für Restaurants in der Nähe von Burgen innerhalb der Schweiz unter Verwendung mehrerer CTEs und räumlicher Distanzfunktionen.

Waldwege

Alle Waldwege einer Gemeinde.

Erläuterung:

- `ST_Union` in boundary: Mehrere matching Polygone würden sonst einen impliziten Cross-Join erzeugen.
- `ST_Union` in forests: Fasst alle Waldgeometrien zusammen → Der EXISTS-Check läuft nur gegen eine Geometrie.
- `MATERIALIZED` in forests: Verhindert, dass der Planner die teure CTE mehrfach auswertet.

- `DISTINCT ON` sorgt dafür, dass jedes Restaurant nur einmal vorkommt.
- `ST_Intersection` erzeugt aus zwei Geometrien nur den überlappenden Teil.

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

-- 1. Gebietsgrenze
WITH boundary AS MATERIALIZED (
  SELECT ST_Union(geom) AS geom
  FROM postpass_polygon
  WHERE tags @> '{"name": "Pfäffikon", "boundary": "administrative"}'
),
-- 2. Waldflächen innerhalb der Grenze
forests AS (
  SELECT ST_Union(p.geom) AS geom
  FROM postpass_polygon p
  JOIN boundary b ON p.geom && b.geom
                 AND ST_Intersects(p.geom, b.geom)
  WHERE p.tags @> '{"landuse": "forest"}'
     OR p.tags @> '{"natural": "wood"}'
),
-- 3. Relevante Wege innerhalb der Grenze (track / path / unclassified)
ways AS (
  SELECT l.osm_id, l.tags, l.geom
  FROM postpass_line AS l
  JOIN boundary b ON l.geom && b.geom
                 AND ST_Intersects(l.geom, b.geom)
  WHERE l.tags @> '{"highway": "track"}'
     OR l.tags @> '{"highway": "path"}'
     OR l.tags @> '{"highway": "unclassified"}'
)
-- 4. Nur der Teil der Wege, der wirklich im Wald liegt
SELECT
  w.osm_id,
  w.tags->>'name'          AS name,
  w.tags->>'highway'       AS highway,
  w.tags->>'smoothness'    AS smoothness,
  w.tags->>'surface'       AS surface,
  w.tags->>'tracktype'     AS tracktype,
  w.tags->>'motor_vehicle' AS motor_vehicle,
  ST_Intersection(w.geom, f.geom) AS geom   -- Schnittgeometrie statt Originalweg
FROM ways w
JOIN forests f ON w.geom && f.geom
              AND ST_Intersects(w.geom, f.geom)
```

Listing: Mehrstufige Postpass-Abfrage zur Ermittlung von Waldwegen innerhalb der Gemeinde Pfäffikon mittels räumlicher CTEs und Geometrie-Schnittoperationen.

Dieses Beispiel zeigt eine typische räumliche Analyse mit Postpass: Zuerst wird ein Gebiet bestimmt, danach werden Waldflächen und Wege gefiltert und schliesslich

nur jene Wegabschnitte ausgegeben, die tatsächlich im Wald liegen.

Ausblick

Die Beispiele decken nur einen kleinen Teil dessen ab, was sich mit Spatial SQL und den weltweiten Daten von OSM realisieren lässt. Bereits mit einfachen SQL-Abfragen lassen sich interessante räumliche Analysen durchführen, etwa:

- Restaurants in der Nähe von Bahnhöfen,
- Waldwege innerhalb bestimmter Gemeinden,
- Burgen entlang von Wanderwegen,
- statistische Auswertungen von OSM-Tags,
- Routing- und Distanzanalysen sowie
- eigene thematische Karten.

Spatial SQL wird zudem häufig zusammen mit anderen Werkzeugen verwendet, beispielsweise mit:

- PostgreSQL/PostGIS
- QGIS
- Python (GeoPandas, Shapely)
- Webkarten mit Leaflet oder MapLibre

Wer tiefer einsteigen möchte, kann sich zusätzlich mit Themen wie räumlichen Joins, Indexierung, Performance-Optimierung, rekursiven CTEs oder Netzwerk-Analysen beschäftigen.

Experimentiere mit eigenen Fragestellungen, passe bestehende Abfragen an und erkunde neue OSM-Tags – genau dadurch entsteht Schritt für Schritt ein besseres Verständnis für Spatial SQL und Geodatenanalysen.

Weiter Beispiele findet man auf der Homepage von Postpass.

Mehr Hintergrund gefällig?

Was ist Postpass?

Postpass ist eine SQL-basierte Schnittstelle für OpenStreetMap-Daten auf Basis von PostgreSQL/PostGIS. Postpass basiert vollständig auf Open-Source-Technologien wie PostgreSQL und PostGIS. Postpass ermöglicht relationale Analysen globaler Rohdaten und stellt eine effiziente Alternative zu Overpass QL dar.

Postpass bietet die volle Leistung einer räumlichen PostgreSQL/PostGIS-Datenbank. Im Vergleich dazu ist Overpass QL eine sehr spezielle Sprache. Hier ein Vergleich:

- Postpass bietet mächtigere Ausdrücke: SQL (mit Postpass) erlaubt komplexe Joins, Window-

Funktionen, CTEs, laterale Abfragen und erweiterte PostGIS-Operationen, die in Overpass QL nur umständlich oder gar nicht möglich sind.

- Etablierte Syntax: Dieselbe Syntax und dieselben Optimierungen laufen potenziell überall, wo PostgreSQL/PostGIS verfügbar ist – im Gegensatz zu Overpass QL.
- Performance: Für viele geometriebasierte Abfragen ist Postpass schneller als Overpass QL, besonders bei Abfragen mit wiederverwendeten Unterabfragen.

Kurz gesagt: Postpass eignet sich für tiefere Geo-Datenanalysen. Overpass QL bleibt die einfachere Wahl für schnelle Ad-hoc-OSM-Abfragen.

Postpass hat aktuell Version 2. Es gibt zwei Versionen von Postpass, eine auf https://overpass-turbo.eu/ und eine auf https://overpass-ultra.us/. Alle Beispiele hier verwenden die Instanz auf overpass-turbo.eu.

Wichtig: Wenn mit Overpass-Turbo gearbeitet wird, benötigt man vor der SQL-Query diese Zeile: `{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}`. Dies gehört nicht zur SQL-Query, sondern ist eine Direktive für Overpass-Turbo und besagt: Wechsle von Overpass QL zu SQL-Modus. Die SQL-Query wird an den Postpass-Server geschickt und das Ergebnis wird als Karte dargestellt.

Im Anhang sind die wichtigsten Postpass-Tabellen zusammengestellt.

Was ist SQL?

SQL steht für Structured Query Language und ist die Standardsprache, um mit Datenbanken zu arbeiten. ("Gib mir Daten", "Speichere das", "Ändere das", "Lösche das")

Wem die Grundlagen von SQL fehlen, dem sei diese "10 easy steps to a complete understanding of SQL". empfohlen.

SQL-Abfragen der Tags

Was sind JSONB-Daten?

In PostgreSQL (und damit auch in Postpass) ist das ein Datentyp, um strukturierte Daten zu speichern. OSM ist nicht klassisch tabellarisch aufgebaut, jedes Objekt hat nicht die gleichen Eigenschaften (Key-Value). Daher ist eine flexible Speicherung als JSONB ist die Lösung. Beispiel (so werden die Informationen in der Spalte Tags abgespeichert):

```
{
    "name": "Pizzeria Roma",
    "amenity": "restaurant",
    "cuisine": "italien"
}
```

Um diese Eigenschaften abzufragen, verwenden wir `->>` und `@>`.

`->>` = Wert aus JSONB lesen (Text / Wert holen) `@>` = Prüft ob JSONB etwas enthält - True / False

(Bedingung prüfen)

Hinweis `tags @> ⋯` ist schneller als `tags->>`, da ersteres den GIN-Index von Postpass verwendet.

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT osm_id, osm_type, tags->>'name' AS "name", ST_PointOnSurface(geom) AS geom
FROM postpass_pointpolygon
WHERE geom && {{bbox}}
  --AND tags @> '{"amenity": "restaurant"}'
  AND tags->>'amenity' = 'restaurant' -- langsamer, da kein Index verwendet
```

Listing: Vergleich einer Postpass-Abfrage mit direktem JSONB-Tag-Zugriff (→>) statt indexoptimiertem @>-Operator.

Spatial SQL

Spatial SQL bietet die Bausteine für die räumliche Analyse. So können standortbasierte Daten mit der Struktur von SQL abgefragt werden, wobei mit Geodatentypen gearbeitet wird.

Typische Anwendungen

- OSM-Auswertungen
- GIS / Kartenanalyse
- Standortanalyse
- Routing/Entfernungen
- Geodaten-Export (GeoJSON, CSV)

Warum Spatial SQL/PostGIS

- komplexe räumliche Analysen
- Joins zwischen mehreren Tabellen
- Distanz, Fläche, Überschneidungen
- Aggregationen und Statistiken
- Wiederholbare Analysen in Datenbanken

Der `ST_`-Prefix bedeutet "Spatial Type" und wurde von Standards wie OGC/SFA und SQL/MM geprägt. Der Prefix ist vor allem sinnvoll als Standardisierungs- und Namespacing-Konvention.

Räumliche Filterung

Es gibt viele Möglichkeiten der räumlichen Filterung. Hier ein paar Varianten:

- Variante A — über Fensterausschnitt
- Variante B — über zwei Eckkoordinaten
- Variante C — entlang einer exakten Grenze

Variante A — Über Fensterausschnitt mit bbox-Platzhalter wie oben.

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT osm_id, tags->'name' AS name, tags, ST_PointOnSurface(geom)
FROM postpass_pointpolygon
WHERE
  tags->>'historic'='castle'
  AND tags->>'name' IS NOT NULL
  AND geom && {{bbox}}
```

Listing: Postpass-Abfrage für Burgen mit Namen im aktuellen Kartenausschnitt.

Variante B — über zwei Eckkoordinaten, mit `ST_MakeEnvelope()`.

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

WITH spatial_extent AS (
  SELECT ST_MakeEnvelope(5.96, 45.82, 10.49, 47.81, 4326) AS geom
)
SELECT osm_id, tags->'name' AS name, tags, ST_PointOnSurface(geom)
FROM postpass_pointpolygon
WHERE
  tags->>'historic'='castle'
  AND tags->>'name' IS NOT NULL
  AND ST_Intersects(geom, (SELECT geom FROM spatial_extent))
```

Listing: Postpass-Abfrage für Burgen innerhalb einer fest definierten Bounding-Box der Schweiz mittels ST_MakeEnvelope().

Wenn du mehr über WITH/CTE-Statements wissen willst, siehe dir dieses Kapitel CTE's an.

Variante C — Entlang einer exakten Grenze (mit `ST_Intersects`)

- Mittels genauer Angabe der `osm_id` und des `osm_type` vom OSM-Element. Das erste Beispiel, mit den Aussichtstürmen in der Schweiz, oben verwendet diese Variante.
- Mittels genauer Angabe des Namens - wobei dieser genau dem Namen des geografischen Orts (Flurname/Gemeinde/Kanton/Land) entsprechen muss. Das Beispiel, mit Waldwegen Pfäffikon ZH, oben verwendet diese Variante.

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}
```

```
WITH spatial_extent AS (
  SELECT geom FROM postpass_polygon WHERE osm_type='R' AND osm_id=51701 -- CH
)
SELECT osm_id, tags->>'name' AS name, tags, ST_PointOnSurface(geom)
FROM postpass_pointpolygon
WHERE
  tags->>'historic'='castle'
  AND tags->>'name' IS NOT NULL
  AND ST_Intersects(geom, (SELECT geom FROM spatial_extent))
```

Listing: Postpass-Abfrage für Burgen innerhalb der exakten Landesgrenze der Schweiz mittels OSM-Relation und ST_Intersects().

APPENDIX: Postpass-Tabellen

Postpass-Tabellen sind spezielle Datenbanktabellen, die auf OSM-Daten basieren und für Abfragen mit PostgreSQL + PostGIS optimiert sind. Statt OSM-Rohdaten (Punkt, Linie, Polygon) direkt zu verwenden, bekommt man fertige Tabellen mit Geometrien und JSON-Tags.

Wie bereits kennen gelernt, hier die Tabelle `postpass_point` mit den gemeinsamen Attributen (`osm_type, osm_id, tags, geom`):

```
CREATE TABLE postpass_point (
    osm_type character(1) NOT NULL,
    osm_id bigint NOT NULL,
    tags jsonb,
    geom geometry(Point,4326)
)
```

Listing: Vereinfachte Tabellenstruktur von postpass_point mit OSM-Tags (JSONB) und Punktgeometrie (geometry(Point,4326)).

`postpass_line`, `postpass_polygon`, `postpass_pointpolygon` sind alle gleich aufgebaut.

Das sind die wohl wichtigsten Tabellen und Views von Postpass: `postpass_point`, `postpass_line`, `postpass_polygon`, `postpass_pointpolygon`.

Tabelle / View	Inhalt	Typische Objekte
`postpass_point`	Punktobjekte	Aussichtstürme, Haltestellen, einzelne POIs
`postpass_line`	Linienobjekte	Strassen, Wege, Flüsse, Bahnlinien
`postpass_polygon`	Flächenobjekte	Gebäude, Wälder, Seen, administrative Grenzen
`postpass_pointpolygon`	Punkt- und Flächenobjekte gemeinsam	Restaurants, Cafés, Schulen, Geschäfte, Burgen

Wann welche Tabelle sinnvoll ist:

- `postpass_point`: Wenn nur punktförmige OSM-Objekte gesucht werden.
- `postpass_line`: Für Wege, Strassen oder andere Linienobjekte.
- `postpass_polygon`: Für Flächenanalysen wie Gebäude, Wälder oder Grenzen.
- `postpass_pointpolygon`: Sinnvoll, wenn Objekte sowohl als Punkt als auch als Fläche vorkommen können (z.B. Restaurants oder Burgen).

Performance-Hinweise:

- Möglichst die spezifischste Tabelle verwenden.
- `postpass_point` ist meist schneller als `postpass_pointpolygon`, weil weniger Geometrien durchsucht werden.
- Kombinierte Views wie `postpass_pointlinepolygon` sind flexibler, aber oft langsamer.
- Räumliche Vorfilter wie `geom && {{bbox}}` verbessern die Performance deutlich.
- Funktionen wie `ST_Intersects()` möglichst mit einem Bounding-Box-Filter (`&&`) kombinieren.

Die weiteren Tabellen sind entweder weniger wichtig (`postpass_pointlinepolygon`, `postpass_pointline`, `postpass_linepolygon`, `middle tables`) oder sollten nicht verwendet werden (compatibility views). Das vollständige Postpass-Schema zum Nachschlagen ist hier.

APPENDIX: PostGIS-Funktionen

Sammlung einiger PostGIS-Funktionen, die oft für Queries verwendet werden:

Geometrie erstellen:

- `ST_MakePoint` — z.B. ST_MakePoint(-71.06, 42.28)
- `ST_MakeEnvelope` — Erstellt ein rechteckiges Polygon aus Koordinaten.
- `ST_AsText` — Nicht mit Postpass, jedoch oft mit PostGIS selber.

Räumliche Beziehungen inkl. Nachbarschaft:

- `ST_Distance(a.geom, b.geom)` — Berechnet den Abstand zwischen zwei Geometrien.
- `ST_DWithin(geom, point, 1000)` — Achtung: projiziertes CRS bzw. GEOGRAPHY verwenden!
- `ST_ClosestPoint(a.geom, point.geom)` — Hat kein Overpass-Äquivalent

Räumliche Beziehungen prüfen (True/False):

- `ST_Intersects(a.geom, b.geom)` — Prüft ob sich zwei Geometrien berühren oder schneiden.
- `ST_Within(a.geom, b.geom)` — Prüft, ob der POI innerhalb der neighborhood liegt.
- `ST_Touches(a.geom, b.geom)` — Hat kein Overpass-Äquivalent

Beim Spatial SQL gibt es verschiedene Funktionen für das räumliche Enthaltensein,

> die ähnlich klingen, wie `ST_Contains`, `ST_Covers`, `ST_Intersects`, und `ST_Within`. Kurz gesagt: Man nehme `ST_Intersects`! (Quelle).

Geometrien messen:

- `ST_Length(geom)` — Berechnet die Länge der Linie.
- `ST_Area(geom)` — Berechnet die Fläche eines Polygons.

Neue Geometrien erzeugen/verändern:

- `ST_Buffer(geom, 100)` — Achtung: projiziertes CRS bzw. GEOGRAPHY verwenden!
  - `ST_Intersection(a.geom, b.geom)` — Gibt nur den gemeinsam überlappenden Teil von a und b zurück.
  - `ST_PointOnSurface(geom)` — Erzeugt einen Punkt, der garantiert auf der Fläche liegt.

Noch Fragen? Wende dich an OpenStreetMap Schweiz oder Stefan Keller!