

Übung 7

Geodatenbanken

1 Einleitung

In dieser Übung werden Sie lernen, wie GIS-Prozesse direkt mit einer Geodatenbank durchgeführt werden können, was insbesondere für grosse Datensätze und (ähnlich wie bei Python) zur Automatisierung von Vorteil sein kann.

Die Kenntnisse, welche Sie in dieser Übung erlangen, sind für das Projekt, aber auch für das generelle Arbeiten mit Geodatenbanken von hoher Wichtigkeit.

! Beim Arbeiten mit GIS und beim Programmieren führt praktisch nie nur eine Lösung alleine zum Ziel. Was wir Ihnen geben, sind Vorschläge. Eigene Lösungen, die zum gleichen Ziel führen, sind ebenso willkommen.

2 Überblick

2.1 Lernziele

- Sie können Informationen abfragen und diese als neue Shapefiles speichern.
- Sie können Manipulationsoperationen nutzen, um einem Datensatz bestimmte (topologische oder geometrische) Informationen hinzuzufügen.
- Sie können Informationen eines Datensatzes zusammenführen.

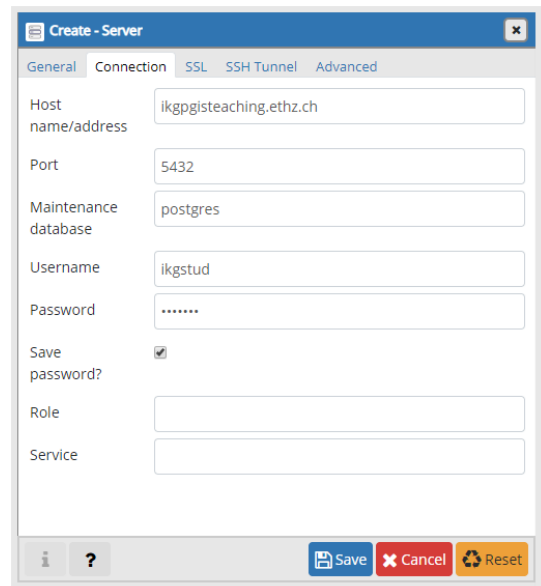
2.2 Daten

Die Daten, die wir heute verwenden, wurden mehrheitlich vom Open Data Portal der Stadt Zürich zur Verfügung gestellt (data.stadt-zuerich.ch). **Die Wanderwege und die Bodenbedeckung dürfen nur innerhalb des GIS GZ-Kurses verwendet werden** (siehe https://geodata4edu.ethz.ch/Infoblatt_Nutzungsbestimmungen_Swisstopo_EN.pdf für allgemeine Nutzungsbestimmungen).

2.3 Datenanalyse mit Geodatenbanken

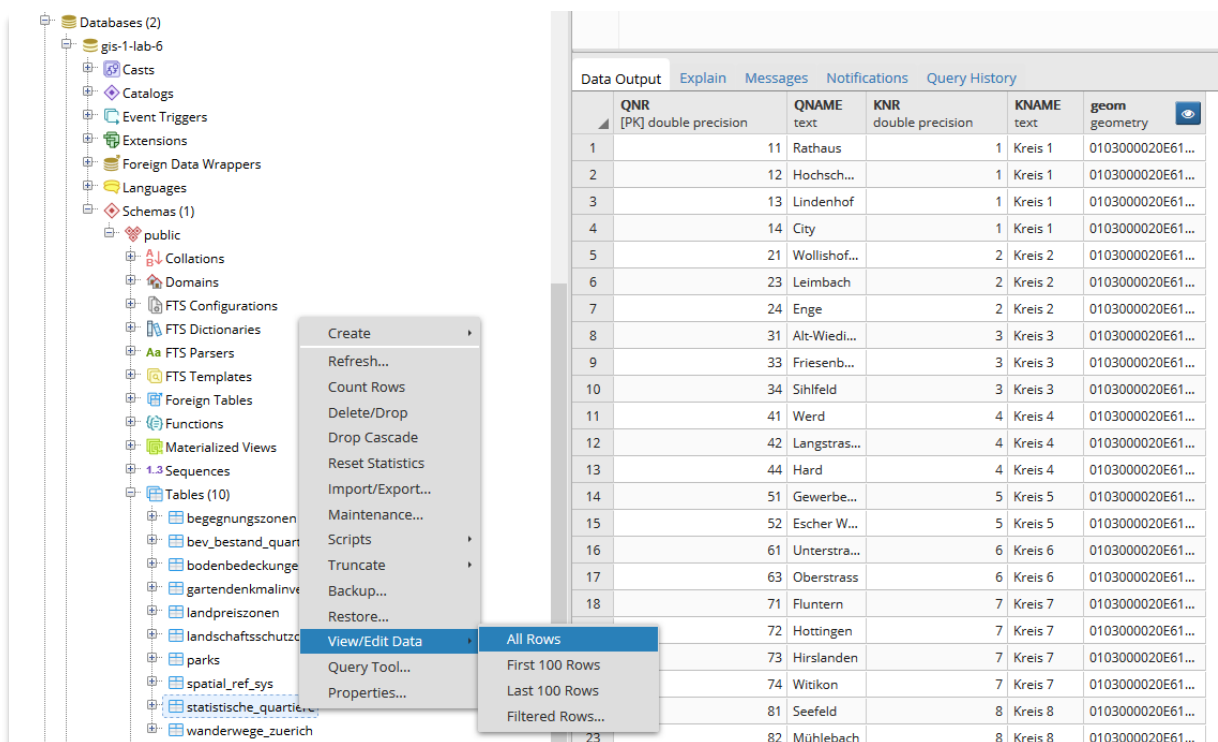
Geodatenbanken wie Oracle Spatial oder PostGIS sind mächtige Werkzeuge, da sie die Funktionen von relationalen Datenbanken mit geographischen Konzepten erweitern, und somit den Gebrauch der Abfragesprache SQL ermöglichen. In SQL geschriebene Anweisungen sind oft sehr kurz und präzise, und werden vom SQL-Compiler optimiert, was insbesondere bei grossen Datensätzen von Vorteil ist.

In diesem Teil der Übung lernen Sie, wie einige der Analysen vom ersten Teil mit SQL durchgeführt werden können. Wir verwenden dazu die PostGIS Datenbank, welche auf der Open Source-Datenbank PostgreSQL basiert. Wir haben für Sie bereits eine Datenbank eingerichtet, auf die Sie mit dem Programm pgAdmin zugreifen können.



Starten Sie dazu zuerst pgAdmin (*Windows Start-Menü > pgAdmin eintippen*). Ein Browser startet. Klicken Sie rechts auf *Servers* und wählen Sie *Create > Server...* um eine neue Serververbindung herzustellen. Sie müssen dazu im ETH-Netzwerk sein (oder per VPN eingeloggt). Geben Sie einen Namen für die Verbindung ein, wechseln Sie zum *Connection* Reiter, und geben Sie bei *Host name/address* **ikgpgisteaching.ethz.ch** ein. Als *username* wählen Sie **ikgstud**, und als Passwort **gis1lab**. Klicken Sie dann auf **Save**.

Öffnen Sie dann das erschienene Baum-Menü, bis Sie *Tables* der Datenbank *gis-1-lab-6* sehen (unten dargestellt). Rechtsklick auf eine Tabelle und *View/Edit Data > All Rows* zeigt den Inhalt der Tabelle, was oft sehr hilfreich sein kann.



	QNR [PK] double precision	QNAME text	KNR double precision	KNAME text	geom geometry
1	11	Rathaus		1 Kreis 1	0103000020E61...
2	12	Hochsch...		1 Kreis 1	0103000020E61...
3	13	Lindenhof		1 Kreis 1	0103000020E61...
4	14	City		1 Kreis 1	0103000020E61...
5	21	Wollishof...		2 Kreis 2	0103000020E61...
6	23	Leimbach		2 Kreis 2	0103000020E61...
7	24	Enge		2 Kreis 2	0103000020E61...
8	31	Alt-Wiedi...		3 Kreis 3	0103000020E61...
9	33	Friesenb...		3 Kreis 3	0103000020E61...
10	34	Sihlfeld		3 Kreis 3	0103000020E61...
11	41	Werd		4 Kreis 4	0103000020E61...
12	42	Langstras...		4 Kreis 4	0103000020E61...
13	44	Hard		4 Kreis 4	0103000020E61...
14	51	Gewerbe...		5 Kreis 5	0103000020E61...
15	52	Escher W...		5 Kreis 5	0103000020E61...
16	61	Unterstra...		6 Kreis 6	0103000020E61...
17	63	Oberstrass		6 Kreis 6	0103000020E61...
18	71	Fluntern		7 Kreis 7	0103000020E61...
	72	Hottingen		7 Kreis 7	0103000020E61...
	73	Hirslanden		7 Kreis 7	0103000020E61...
	74	Witikon		7 Kreis 7	0103000020E61...
	81	Seefeld		8 Kreis 8	0103000020E61...
	82	Mühlebach		8 Kreis 8	0103000020E61...

2.3.1 Daten und Datenmodell

Für diese Übung haben wir die Shapefiles in die PostGIS Datenbank überführt. Wir zeigen hier exemplarisch nur einen Ausschnitt der entsprechenden Datenbankdefinition in der Data Definition Language (DDL; ein Teil von SQL), die wir für die Erstellung gebraucht haben.

```
CREATE TABLE statistische_quartiere
(
    "QNR" double precision,
    "QNAME" text,
    "KNR" double precision,
    "KNAME" text,
    geom geometry(Polygon,4326),

    CONSTRAINT statistische_quartiere_pk PRIMARY KEY ("QNR"),
    CONSTRAINT statistische_quartiere_pk_unique UNIQUE ("QNR")
);

CREATE TABLE bev_bestand_quartiere
(
    "StichtagDatJahr" bigint,
    "QuarSort" bigint,
    "QuarLang" text,
    "AnzBestWir" bigint,

    CONSTRAINT bev_bestand_quartiere_pk PRIMARY KEY ("QuarSort"),
    CONSTRAINT bev_bestand_quartiere_pk_unique UNIQUE ("QuarSort"),
    CONSTRAINT bev_bestand_quartiere_fk FOREIGN KEY ("QuarSort")
        REFERENCES statistische_quartiere ("QNR") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
);
```

Der **Primary Key** dient dazu, die Einträge einer Tabelle eindeutig zu identifizieren und auf sie zuzugreifen.

Der **Foreign Key** ist ein Attribut, welches auf einen Primärschlüssel einer anderen Relation verweist.

Der **Index** wird für jede Tabelle angelegt um die Suche und das Sortieren nach bestimmten Feldern der Tabelle zu beschleunigen.

2.3.2 Thematische SQL-Abfragen

Die Sprache SQL dient dazu, Datenbankabfragen zu formulieren. Die wichtigsten SQL-Befehle lauten:

DESCRIBE	Anzeige des Schemas einer Tabelle
SELECT	Abfrage von Daten aus einer Tabelle
INSERT INTO	Daten in Tabelle einfügen
UPDATE	Bestehende Daten in Tabelle ändern
DELETE	Daten aus Tabelle löschen (Schema bleibt erhalten)

Dem Datenbankbenutzer, den Sie für diese Übung verwenden, wurden keine Rechte für die Änderung der Daten, sondern lediglich lesender Zugriff gegeben. Für diese Übung ist daher nur der SELECT-Befehl relevant, der im Folgenden näher erklärt wird.

Eine SQL - Abfrage gliedert sich in der Regel in 4 Teile:

1. Welche Spalten werden abgefragt?	SELECT
2. Auf welche Tabellen beziehen sich die Spalten?	FROM
3. Welche Bedingungen bestehen?	WHERE ... AND/OR
4. Wie sollen Tabellen verknüpft werden? (JOIN)	<i>Join-Bedingung;</i>

Beispiel: Finden Sie die Ids (Nummern) aller Quartiere heraus, deren Anzahl Einwohner kleiner als 5000 ist! Zur Lösung dieser Aufgabe müssen die beiden Tabellen *statistische_quartiere* und *bev_bestand_quartiere* und miteinander verknüpft werden.

1. Welche Spalten werden abgefragt?	SELECT Q."QNR", Q."geom"
2. Auf welche Tabellen beziehen sich die Spalten?	FROM statistische_quartiere Q, bev_bestand_quartiere B
3. Welche Bedingungen bestehen?	WHERE B."AnzBestWir" < 5000
4. Wie sollen Tabellen verknüpft werden? (Join)	AND B."QuarSort"=Q."QNR"

Achtung: Wir müssen Spaltennamen **Anführungszeichen verwenden ("...")**, weil SQL sonst alles als Kleinbuchstaben interpretiert. Im Gegensatz zu Python werden Anführungszeichen bei SQL als Variablen mit Gross- / Kleinschreibung interpretiert, während einfache **Hochkommas ('...')** für **Strings** verwendet werden. Hier haben wir für die Datenbanktabellen die gleichen Attributnamen wie in den Shapefiles verwendet. Wenn man eine PostGIS-Datenbank

von Grund auf neu aufsetzt, lohnt es sich aber alle Namen konsequent klein zu schreiben, damit keine Anführungszeichen benötigt werden.

Hinweise: Im FROM-Teil des SQL-Statements werden in diesem Beispiel Abkürzungen für die Tabellennamen eingeführt. Diese können dann im SELECT- und WHERE Teil in der Form *Abkürzung.Spaltenname* benutzt werden.

Aufgabe 1

Um ein SQL Statement auszuführen müssen sie im linken Fenster auf die Datenbank *gis-1-lab-6* rechtsklicken, und *Query Tool...* auswählen. Im erscheinenden Fenster können Sie dann die SQL Statements eingeben und mit einem Klick auf *Execute / Refresh (F5)* ausführen.

➔ Führen Sie das Beispiel aus. Wie viele Quartiere haben mehr als 25'000 Einwohner?

Tipp: Klicken Sie in der resultierenden Geometriespalte oben auf das Auge. Sie können dann wie in ArcGIS eine Visualisierung der Geometrien sehen!

Aufgabe 2

In der vorigen Aufgabe haben Sie mit einem *impliziten Join* gearbeitet. Bei diesem werden einfach zwei Attribute von zwei Tabellen in einem WHERE-Statement gleichgesetzt (z.B. B."QuarSort"=Q."QNR"). Der Vorteil dabei ist natürlich, dass dies sehr unkompliziert mit anderen WHERE-Statements kombiniert werden kann.

Es gibt aber auch *explizite Joins* in SQL, welche die folgende Form haben. Beide Abfragen werden gleich in Maschinencode übersetzt und so ist auch das Ergebnis gleich.

```
SELECT * FROM statistische_quartiere Q
INNER JOIN bev_bestand_quartiere B ON B."QuarSort"=Q."QNR";
```

➔ Versuchen Sie, mit dem expliziten Join alle Quartiere, die mehr als 25'000 Einwohner haben, zu finden. Sie können nach dem expliziten Join genau gleich ein WHERE-Statement anhängen.

Aufgabe 3

Wir können nun beginnen, Anfragen direkt auf der Geodatenbank auszuführen. Achten Sie auf die richtige Verwendung von Anführungszeichen und Hochkommas.

➔ Finden Sie von den Bodenbedeckungsflächen alle Wälder (BESCHREIB ist Wald oder Wald offen) die grösser als 3 km² sind.

2.3.3 Räumliche SQL-Abfragen

Um räumliche Abfragen stellen zu können, benutzen wir die SQL-Erweiterung PostGIS. Diese Erweiterung beinhaltet verschiedene räumliche Datentypen (*geometry* als abstrakten Typ für alle Geometrien, dann aber auch *Point*, *LineString*, *Polygon*, etc.¹) sowie geometrische Operationen, die auf diesen Datentypen ausgeführt werden können. Nachfolgend werden sowohl einige der Datentypen, als auch einige der Operationen etwas näher erklärt.

OpenGIS WKT (Well-Known Text)

Ein Grossteil der PostGIS Datentypen kann im Well-Known Text Format spezifiziert werden:

- POINT(0 0)
- LINESTRING(0 0,1 1,1 2)
- POLYGON((0 0,4 0,4 0,0 0),(1 1, 2 1, 2 2, 1 2,1 1))

Dabei werden die Koordinaten jeweils in Listenform nach dem Geometrietyp geschrieben. Achten Sie darauf, dass Koordinatenpaare kein Komma dazwischen haben, die einzelnen Koordinaten aber durch Kommas getrennt sind. Bei Polygonen ist die erste Koordinate gleich der letzten. Weiter können bei einem Polygon «Löcher» angegeben werden, welche als weitere Polygone auf das erste folgen. In PostGIS können Sie einen neuen Tabelleneintrag wie folgt aus WKT generieren. Hierbei fügt der Befehl INSERT INTO neue Einträge zu einer Tabelle hinzu, welche die nach VALUES angegebenen Werte enthalten:

```
INSERT INTO table_name (geometry, name_attribute)
VALUES (ST_GeomFromText('POINT(8.50 47.40)', 4326), 'Hoenggerberg');
```

Achtung: In PostGIS werden Geometrien generell als «flach» bzw. projiziert angenommen (also in einem kartesischen, und nicht etwa sphärischen Koordinatensystem). Für Berechnungen auf der Erdoberfläche empfiehlt sich ein Blick auf den *Geography*-Typ². Nichtsdestotrotz können Sie ein Koordinatensystem angeben (oben via der *SRID* in der Funktion ST_GeomFromText(...)), das dann auch bei der Umwandlung in einen *Geography*-Typ berücksichtigt wird.

¹ https://postgis.net/docs/using_postgis_dbmanagement.html#RefObject

² https://postgis.net/docs/using_postgis_dbmanagement.html#PostGIS_Geography

Abfragemodelle in Geodatenbanken

Geodatenbanken nutzen normalerweise ein zweistufiges Abfragemodell um räumliche Abfragen und räumliche Joins zu bearbeiten. Es werden zwei verschiedene Operationen ausgeführt um am Ende das exakte Ergebnis zu erhalten.

Die beiden Operationen werden als Primär- und Sekundärfilter Operationen bezeichnet.

- **Primärfilter:** Erlaubt schnelle Selektion aller Datensätze, die dann an den Sekundärfilter weitergegeben werden. Der Primärfilter vergleicht nur geometrische Näherungswerte um den Rechenaufwand zu reduzieren. Oft werden nur die Begrenzungsrahmen (Bounding-Box) verglichen. Das Ergebnis liefert eine Obermenge der eigentlichen Ergebnismenge.
- **Sekundärfilter:** Wendet exakte Berechnung auf die Ergebnismenge des Primärfilters an und liefert das genaue Ergebnis. Der Filter benötigt mehr Rechenaufwand, aber die genaue Berechnung wird nicht mehr auf alle Datensätze angewandt, sondern nur auf die durch den Primärfilter schon eingeschränkte Menge.

PostGIS nutzt einen räumlichen Index (Generic Index Structure oder R-Baum), um den Primärfilter zu implementieren, d.h. beim Primärfilter wird nur auf der Indextabelle gearbeitet. Der Sekundärfilter ist für die Bestimmung der räumlichen Beziehung zwischen Objekten, basierend auf der geometrischen Lage, zuständig.

Geometrische Operationen auf Geometrieattributen

Eine einfache Primärfilterabfrage wird in PostGIS mit dem **&&** Operator durchgeführt.

Beispiel: `SELECT * FROM statistische_quartiere Q, begegnungszonen B
WHERE Q.geom && B.geom`

Da lediglich der Primärfilter ausgeführt wird, ist diese Funktion zwar schnell, liefert aber auch keine exakten Ergebnisse. Die Ergebnismenge kann auch Objekte enthalten, die selbst keinen räumlichen Zusammenhang haben, wohl aber ihre umgebenden Rechtecke.

Für Sekundärfilterabfragen stellt PostGIS eine Auswahl an Funktionen zur Verfügung³, z.B.:

- `ST_Intersects(geometry_a, geometry_b)`: Prüft, ob zwei Geometrien sich überlappen.
- `ST_Within(geometry_a, geometry_b)`: Prüft, ob Geometrie A komplett innerhalb Geometrie B ist.
- `ST_Area(geometry)`: Berechnet die Fläche eines Polygons im Koordinatensystem des Geometrieattributs (wenn hier Meter o.Ä. gewünscht sind, empfiehlt sich wiederum der *Geography*-Datentyp).
- `ST_Buffer(geometry, radius)`: Berechnet einen Puffer um die Geometrie.
- `ST_Intersection(geometry_a, geometry_b)`: Berechnet den Verschnitt zwischen zwei Geometrien.

³ <https://postgis.net/docs/reference.html>

Aufgabe 4

Sie können *SELECT*-Statements einfach verschachteln, um die Ergebnistabelle in einem nächsten *SELECT*-Statement wieder zu verwenden. Im folgenden wird das Ergebnis des inneren *SELECT*-Statements in der Variable *AL* zwischengespeichert. Beginnen Sie, indem Sie zuerst das innere *SELECT*-Statement vervollständigen (und fehlerfrei zum Laufen bringen), und es dann ins äussere einbauen:

```
SELECT * FROM
    (SELECT PA.* FROM parks PA, gartendenkmalinventare GA
     WHERE ...) AS AL,
    landpreiszonen LP
WHERE ...
```

- ➔ Selektieren Sie die Pärke, welche innerhalb eines Gartendenkmalinventars liegen (inneres Statement).
- ➔ Führen Sie dann mit diesem Layer einen Spatial Join mit den Landpreiszonen durch, um den Landpreiszonentyp an die Pärke zu joinen (äusseres Statement).

Wieviele Parks an exklusiver / guter Lage erhalten Sie?

Aufgabe 5

Wie vorhin angesprochen ist es für «geographische» Berechnungen oft sinnvoll, die Geometrie-Kolonnen in *Geography*-Typen umzuwandeln, damit Sie mit Metern statt Grad arbeiten können (die Geometrien sind in dieser Übung in WGS84 gespeichert). Sie können das einfach machen, indem Sie `geom::geography` statt `geom` schreiben. Das sieht dann z.B. so aus:

```
SELECT * FROM statistische_quartiere Q, begegnungszonen B
WHERE Q.geom::geography && B.geom::geography
```

- ➔ Erzeugen Sie mit der Funktion `ST_Buffer` um die Limmat und die Sihl herum einen Puffer von 250 Metern Radius (beide sind in der Tabelle *landschaftsschutzobjekte*, identifiziert durch "NUMMER"='KSO-25' und "NUMMER"='KSO-31'). Benutzen Sie folgendes Gerüst:

```
SELECT ST_Buffer(geom::geography, ...) FROM landschaftsschutzob-
jekte
WHERE ...
```

- ➔ Wir müssen nun die Puffer «dissolvern», d.h. in ein einzelnes Polygon zusammenführen. In PostGIS geschieht das mit dem Befehl `ST_Union(...)`. Dieser kann einfach um den Puffer «gelegt» werden, wobei wir wieder zurück zum Geometrie-typ müssen:

```
ST_Union(ST_Buffer(geom::geography, ...))::geometry)
```

Tipp: Schauen Sie sich das ganze unbedingt zwischendurch mittels dem «Auge» beim Spaltennamen auf einer Karte an!

- ➔ Benutzen Sie zum Schluss die Funktion `ST_Difference(geometry_a, geometry_b)`, um diese Pufferfläche von allen Begegnungszonen zu entfernen. Dazu müssen Sie wieder ein *SELECT*-Statement verschachteln. Diese Abfrage verwendet einen Sekundärfilter. Dabei wird zuvor intern der Primärfilter ausgeführt. PostGIS entfernt leere Polygone nicht ohne weiteres. Mit einem geeigneten *WHERE*-Statement können Sie das aber leicht lösen. Da PostGIS die SQL-Anfrage automatisch optimiert, spielt es auch keine Rolle, wenn Sie die `ST_Difference` mehrere Male berechnen:

```
SELECT ST_Difference(...) FROM  
...  
WHERE NOT ST_IsEmpty(ST_Difference(  
    begegnungszonen.geom, puffer.st_union))
```

Sie sollten das folgende Ergebnis erhalten:

