NETGIS
Gesellschaft für
Geoinformation und Umweltplanung

Tutorial

„Einführung in PostgreSQL mit PostGIS unter MS Windows“

Empfehlung von weiterführender Literatur:

- PostgreSQL Administration, Peter Eisentraut & Bernd Helmle, O'Reilly Verlag 2013
- PostgreSQL 8.4: Das Praxisbuch, Thomas Pfeiffer & Andreas Wenk, Galileo Computing 2009
- Beginning Databases with PostgreSQL, Neil Matthew & Richard Stones, Apress 2005
- WebMapping mit Open Source-GIS-Tools, Tyler Mitchel & Astrid Emde & Arnulf Christl, O'Reilly Verlag 2008
- PostGIS in Action, Regina O.Obe & Leo S. Hsu, Manning 2011
- Postgis Cookbook, Paolo Corti & Thomas J Kraft, Packt 2014

Online Dokumentationen:
Postgres: http://www.postgresql.org/docs/
PostGIS: http://postgis.refractions.net/documentation/

Inhalt

1. Vorbemerkung

PostgreSQL ist eine hoch entwickelte objektrelationale Open Source Datenbank.
Es wird zu Recht als "das fortschrittlichste Open-Source-Datenbanksystem" bezeichnet. Das System zeichnet sich durch Sicherheit, Datenintegrität und eine ständige Weiterentwicklung durch ein internationales Firmenkonsortium aus.
PostGIS ist die räumliche Erweiterung von PostgreSQL zur Speicherung und Verwaltung von Geodaten. Damit ist der Einsatz von PostGIS als Datenbank-Backend für GIS-Applikationen möglich. Das Projekt implementiert die Simple Feature Access-Spezifikation des Open Geospatial Consortium und wird von der Open Source Geospatial Foundation betreut.

Dieses vorliegende Tutorial wurde unter Verwendung der Versionen von PostgreSQL 9.3 mit der räumlichen Erweiterung Postgis 2.1 erstellt und getestet.
Alle Befehle und SQL Anweisungen sollten auch aufwärts in aktuellen Datenbankversionen kompatibel sein.
Bei neueren Versionen sollte die Installation unwesentlich abweichen.
Die Installation wurde unter den Betriebssystemen MS Windows 7, 10 und MS Windows 2008, 2012 und 2016 Server getestet.

2. Installation von Postgresql Version 9.3.x und Postgis auf Windows Systemen

Die aktuellen Windows-Versionen (Installer und Binary-Pakete) für PostgreSQL stehen hier zum Download bereit: http://www.postgresql.org/download/windows
Wenn Sie die Installer der Firma Enterprisedb herunterladen und benutzen wollen, müssen Sie sich vorher registrieren und einloggen.

2.1 Installation PostgreSQL

1. Installer (z.B. postgresql-9.3.21-1-windows-x64.exe) starten und auf "Next" klicken

2. Zielverzeichnis von PostgreSQL wählen (Standard C:\Program Files\PostgreSQL\9.3) und weiter dem Dialog folgen.

3. Datenverzeichnis wählen (Standard C:\Program Files\PostgreSQL\9.3\data). Hier sollte man ein Verzeichnis wählen, welches regelmäßig einer Sicherung unterliegt.

4. Passwort für den Superuser postgres vergeben. Hier sollte für den Produktivbetrieb unbedingt ein sicheres Passwort verwendet werden.

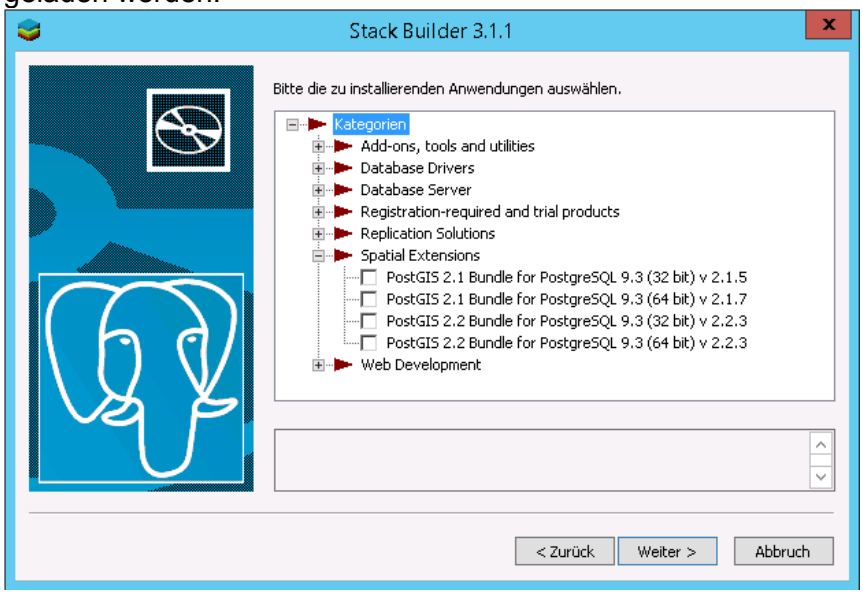
5. Port wählen, auf dem die Datenbank später erreichbar sein soll. Als Standard wird in der Regel der Port 5432 genutzt, es kann z.B. bei der Installation von verschiedenen Versionen aber auch ein anderer Port verwendet werden.

6. Ländereinstellungen (locale) wählen, benutzen Sie ein deutsches Betriebssystem können Sie die Standardeinstellung [Default locale]übernehmen.

7. noch einmal auf next und die Installation beginnen. Das dauert in der Regel nicht länger als eine Minute. Nach erfolgreicher Installation erscheint ein Dialog um den Stack Builder zu starten um zusätzliche Software zu installieren. Sie können z.B. darüber direkt Postgis herunterladen und anschließend installieren (vgl. b Installation Postgis).

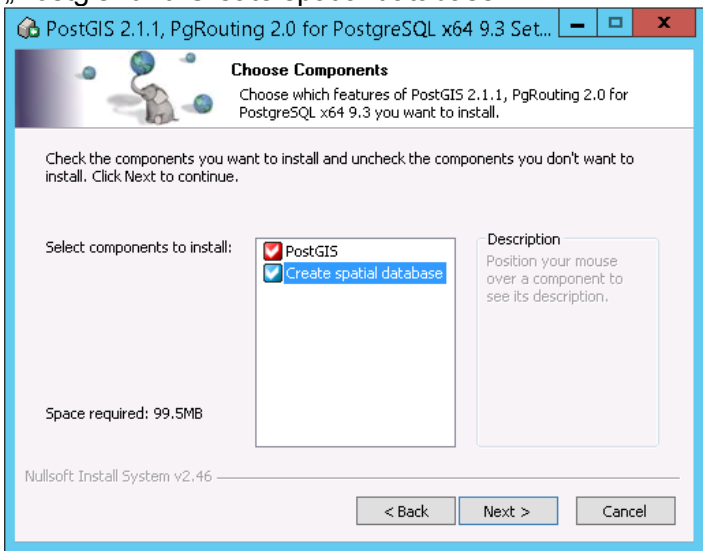
2.2 Installation Postgis

1. Bei weiterer Installation aus dem PostgreSQL-Dialog kann an dieser Stelle POSTGIS auch über die Option „Launch Stack Builder“ über das Internet geladen werden.



Sie können den Installer für Postgis auch downloaden und direkt ausführen (Empfehlung bei der Installation auf Servern, die aus Sicherheitsgründen den Download über das Internet nicht zulassen)
Starten Sie nun den Installationsfile für Postgis (z.B. postgis-bundle-pg93x64-setup-2.1.1-1.exe)

2. Nach der Zustimmung der Lizenzbestimmungen, wählen Sie die Komponenten „Postgis“ und Create spatial database“



3. Im Dialog „choose install location“ empfiehlt sich das Standardverzeichnis unter PostgreSQL

4. Im Dialog „Database Connection“ geben Sie die aus der PostgreSQL Installation vergebenen Zugangsdaten zum Superuser postgres und den Port des Datenbankservers ein.

5. Nach dem anschließenden Dialog „databasename“ bei dem die Standarddatenbank für Postgis geändert werden kann ist die Installation abgeschlossen. Die Frage der Registrierung der GDAL_DATA Umgebungsvariable kann in der Regel bestätigt werden, sofern keine andere Software diese auf dem Rechner bereits benötigt.

2.3 Optimierung des Datenbankservers

Die Voreinstellungen einer Standardinstallation von PostgreSQL sind für die Anwendung in einem Produktivsystem nicht ratsam da die Einstellungen eher für den parallelen Betrieb in einer Entwicklungsumgebung oder für Systeme mit schwacher Hardwareausstattung konzipiert wurden. Es sollten deshalb einige Anpassungen in der Konfiguration vorgenommen werden.
In diesem Zusammenhang empfehlen wir die Lektüre "PostgreSQL Administration" von Peter Eisenkraut aus dem O'Reilly Verlag.
Alle Parameter zur Optimierung können mit einem Editor in der zentralen Konfigurationsdatei **postgresql.conf** vorgenommen werden. Bei einer Standardinstallation befindet sich diese Datei unter Windows in dem Verzeichnis *C:\Program Files\PostgreSQL\9.3\data.* Die Beschreibung der optimalen Einstellungen der Parameter würde den Rahmen des Tutorials sprengen, es sollen deshalb lediglich die Parameter aufgelistet werden, die vor einer Inbetriebnahme angepasst oder überprüft werden sollten.

Einstellungen zur Verbindungskontrolle:
→ listen_adresses (vgl. Kap. Zugangskontrolle)
→ max_connections (maximale Anzahl von Verbindungen die eine Instanz des DB-Servers gleichzeitig offen haben kann)
→ ssl (aktiviert die SSL-Unterstützung des DB-Servers).

Einstellungen zur Speicherverwaltung:
→ shared_buffers (legt die Größe des Shared-Buffer-Pools einer DB-Instanz fest)
→ work_mem (Obergrenze des zur Verfügung stehenden Arbeitsspeichers für DB-Operationen wie Sortieren oder Verknüpfungsalgorithmen)
→ maintenance_work_mem (Obergrenze des zur Verfügung stehenden Arbeitsspeichers für Verwaltungsoperationen zur Erzeugung oder Änderung von DB-Objekten)
→ max_fsm_pages

Einstellungen zur Wartung, Planeinstellungen und Logging:
→ wal_buffers
→ checkpoint_segments
→ checkpoint_timeout
→ effective_cache_size
→ log_line_prefix

Automatische Optimierung mit Zusatztool:
Eine schnelle und gute Analyse des Servers gelingt mit dem kostenlos verfügbaren Tools **EDB Tuning Wizard** der Firma EnterpriseDB.
Man kann es über den Application Stackbuilder in PostgreSQL installieren wenn der Server es zulässt. Es muss allerdings vorher einen Account bei Enterprisedb anlegt werden.
Der Wizzard ist eigentlich selbsterklärend und erzeugt nach entsprechenden Voreinstellungen (Server-Utilization: Development, Mixed-Mode oder Dedicated) eine neue Konfigurationsdatei postgresql.conf. Die Änderungen werden dabei gekennzeichnet und die Originaldatei in einer Kopie gespeichert. Am Schluss muss lediglich der Datenbankdienst neu gestartet werden damit die neue Konfiguration wirksam wird.

2.4 Zugangskontrolle - Freigabe in einem Netzwerk

Nach der Installation ist die Datenbank lediglich vom lokalen Server erreichbar. Zur Freigabe der Datenbank in einem Netzwerk für andere Hosts müssen folgende Einstellungen vorgenommen werden (einfaches Beispiel):

postgresql.conf

Hier muß der Parameter „listen_addresses“ angepasst werden.
Der Parameter konfiguriert die IP-Adressen (oder Adressbereiche) auf denen Datenbankverbindungen entgegen genommen werden. Es können IP-Adressen oder Hostnamen eingestellt werden. Normalerweise wird die Voreinstellung so geändert, dass Verbindungen von allen Adressen (Platzhalter *) möglich sind:

listen_addresses = '*'

Andere Einstellungen siehe Dokumentation.

pg_hba.conf

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
   host          all                 all        192.168.178.8/32         md5
```

In diesem Fall erhält ein einzelner Host über Passwortabfrage die Berechtigung für alle Datenbanken und User, andere Einstellungen siehe Doku.

Bitte beachten Sie die entsprechende Portfreigabe in der Firewall des Servers!

Außerdem sollte nach den Änderungen der PostgreSQL-Dienst neu gestartet werden oder über das Tool Start-> Programme -> PostgreSQL 9.3 -> Reload Configuration.

3. Administration der Datenbank

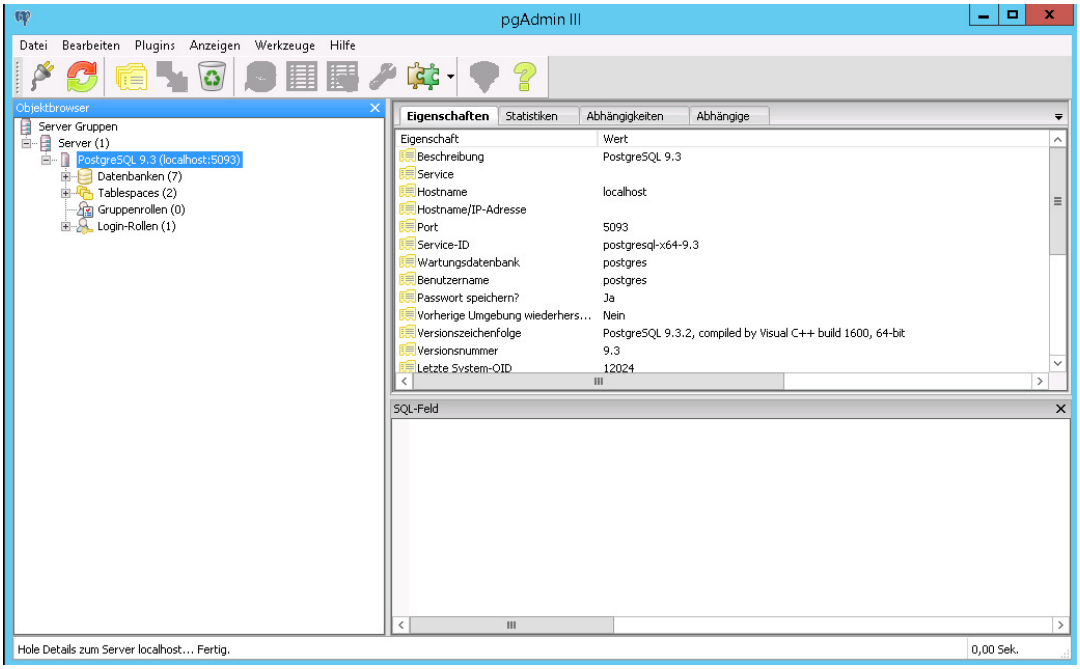
Zur Administration stehen nach der Standardinstallation unter Windows folgende Datenbank-Clients zur Verfügung:

3.1 Terminalprogramm psql

Für Freunde der Kommandozeile kann die Administration des Datenbankservers über den PostgreSQL-Terminal **psql** vorgenommen werden. Sie öffnen das Kommandozeilenprogramm über *Start/Programme/PostgreSQL9.3/SQL Shell (psql)*
Nach dem Start der Konsole melden Sie sich als Administrator „postgres“ an und verbinden sich mit der gewünschten Datenbank, danach können Sie sofort SQL-Statements abschicken.
Sind Sie über psql auf einer Datenbank angemeldet muss jeder Befehl entweder mit /g oder einem Semikolon beendet werden.

3.2 pgAdmin III

Dieser Client ist eine graphische Benutzeroberfläche zur Administration von PostgreSQL.
Sie starten das Programm über *Start/Programme/PostgreSQL9.3/pgadmin III*
Nach der Anmeldung sind die Datenbanken und deren Objekte in einer Baumstruktur im linken Fenster verfügbar. Es können alle Attribute und SQL-Befehle über diese Oberfläche komfortabel einstellen und abschicken.
Die folgende Referenz bezieht sich auf die Arbeit mit dem pgAdmin.
Alle Operationen können natürlich auch mit dem Terminalprogramm auf Komandozeilenebende vorgenommen werden.



Hinweis: In der Version PostgreSQl 9.3 wird als Standard der PG Admin 3 ausgeliefert. Es gibt mittlerweile eine neue Version 4, die natürlich nachinstalliert werden kann und abwärtskompatibel mit der Datenbankversion ist. Hier ist die Oberfläche ähnlich aber ein wenig komplexer und moderner. Weitere Informationen und die Download Resourcen finden Sie hier: https://www.pgadmin.org/

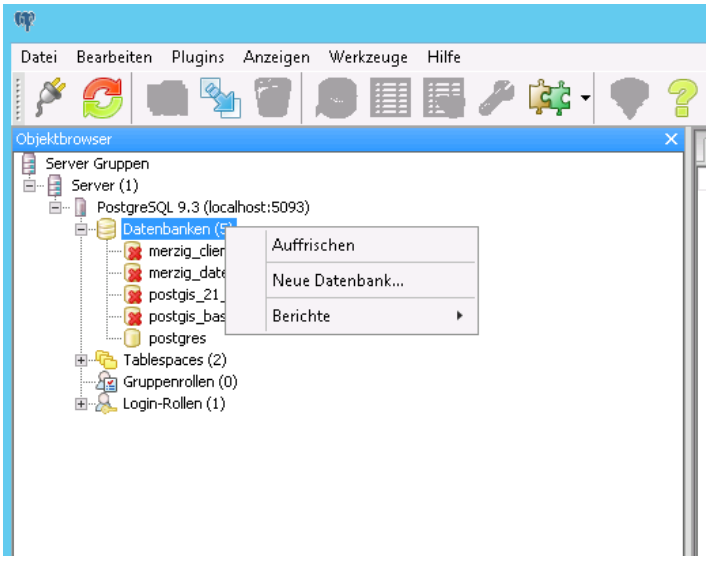
3.3 Anlegen einer neuen "räumlichen" Datenbank mit dem pgAdmin 3

Bei der Installation von PostgreSQL wird automatisch eine Datenbank angelegt: *postgres*
Die Datenbank *postgres* ist die Standardvorlage zum Erstellen von neuen Datenbanken. Diese Datenbank sollte deshalb nicht gelöscht werden.

Nach der Installation von Postgis wird je nach Version ebenfalls eine zusätzliche Datenbank angelegt, z.B. *postgis_21_sample*.
Von dieser Datenbank könnte man nun ein Backup erzeugen (rechte Maustaste auf Datenbank -> Sicherung) und danach ggfls. für seine eigenen Bedürfnisse anpassen und umbenennen (rechte Maustaste auf Datenbank -> Eigenschaften -> Name ändern).

Die Beispieldatenbank beinhaltet alle Funktionen und Referenztabellen die für eine "räumliche" Datenbank benötigt werden. Zudem beinhaltet sie die Tabelle "spatial_ref_sys". Diese Tabelle speichert die Informationen zu den verschiedenen Projektionen. Die Informationen werden z.B. auch für das Umwandeln von Daten in unterschiedliche Projektionssysteme verwendet.

Zur Erzeugung einer neuen Datenbank klickt man im pgAdmin mit der rechten Maustaste auf "Datenbanken" und dann auf "Neue Datenbank..."



Sie müssen im Dialog nun im Reiter "Eigenschaften" einen *Namen* für die Datenbank vergeben und den *Eigentümer* (in unserem Fall "postgres") wählen.
Im Reiter "Definition" wird die *Kodierung* UTF8 empfohlen, zur Erstellung einer neuen „räumlichen“ Datenbank sollte als *Vorlage* die Datenbank als postgis_21_sample verwendet werden.

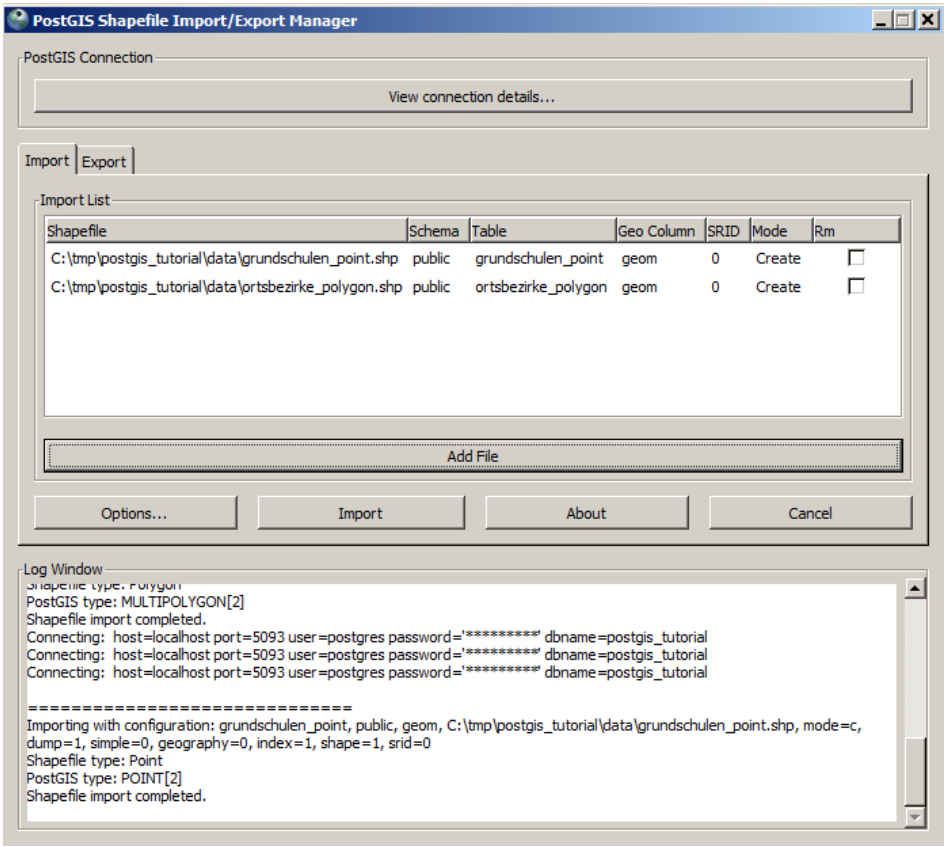
Alle weiteren Einstellungen sind an dieser Stelle nicht relevant und können unverändert bleiben.
Weitere Informationen zur Bedienung des pgAdmin 3 finden Sie hier:
https://www.pgadmin.org/docs/pgadmin3/1.22/

4. Import und Export von Geodaten

Nachdem eine "räumliche" Datenbank angelegt wurde gibt es verschiedene Wege diese mit Geodaten zu befüllen.
Dabei können Kommandozeilen Werkzeuge zum Einsatz kommen oder grafische Oberflächen bzw. externe Programme.

4.1 Datenaustausch mit dem PostGIS Shapefile Import/Export Manager

Den PostGIS Shapefile Import/Export Manager erreichen Sie über den pgAdmin im Menü "Plugins" -> Postgis Shapefile and dbf Loader.
In dem Dialog können Sie einfach Shapefiles hinzufügen und direkt auf die im pgAdmin aktuell ausgewählte Datenbank laden.
Ein häufiges Problem in diesem Zusammenhang ist die Encodierung der .dbf Tabelle des Shapefiles. Da die Zieldatenbank in der Regel in UTF-8 encodiert ist kommt es bei einem in LATIN1 encodierten Shapefile zu einer Fehlermeldung und der Import wird abgeprochen.
Sie können dann einfach bei den Optionen die Quellencodierung Standard UTF-8 auf LATIN1 ändern und der Import wird funktionieren.
Ebenso können Sie über den Mode auswählen, ob eine neue Tabelle angelegt wird oder z.B. die Daten an eine bestehende Tabelle angehängt werden sollen (append).
Das Plugin ermöglicht nur den Import von ESRI Shapefiles und baut auf den folgenden Kommandozeilenprogrammen **shp2pgsql.exe** und **pgsql2shp.exe** auf.
Hiermit können Sie noch umfangreichere Einstellungen machen, und natürlich wiederkehrende Aufgaben z.B. über die Aufgabenplanung erledigen.



4.2 Hilfsprogramme (Loader und Dumper)

Nach der Standardinstallation von PostGIS liegen im bin-Verzeichnis von PostgreSQL die zwei Kommandozeilenprogramme **shp2pgsql.exe** und **pgsql2shp.exe** mit denen es möglich wird PostGIS-Tabellen in das Shape-Format zu konvertieren und umgekehrt diese zu laden.
Tipp: Für eine komfortable Bearbeitung ohne Angabe von Programm-Pfaden empfehlen wir das Anlegen einer Systemvariablen auf das bin-Verzeichnis von PostgreSQL. Dazu gehen Sie über die Systemsteuerung auf System→ Erweiterterte Systemeinstellungen → Umgebungsvariablen. Danach wählen Sie unter den Systemvariablen „Path“ und fügen den Pfad zum bin-Verzeichnis (z.B. C:\Program Files\PostgreSQL\9.3\bin) durch ein Semikolon getrennt hinzu.
Sie können nun die Kommandozeilenprogramme von jedem Ort auf dem Rechner ohne Angabe des Pfades aufrufen.

shp2pgsql

Mit diesem Programm können sehr komfortabel ESRI-Shape-Daten nach Postgis geladen werden. Folgende Optionen stehen zur Verfügung:

-d Löscht die Datenbanktabelle, bevor eine neue Tabelle mit den Daten in der Shape-Datei erstellt wird.
-a Fügt Daten aus der Shape-Datei in die Datenbanktabelle ein. Beachten Sie, dass diese Option zum Laden mehrerer Dateien verwendet werden kann, wenn die Dateien dieselben Attribute und Datentypen haben müssen.
-c Erstellt eine neue Tabelle und füllt sie aus der Shape-Datei. Dies ist der Standardmodus.
-p Erzeugt nur den SQL-Code für die Tabellenerstellung, ohne die eigentlichen Daten hinzuzufügen. Dies kann verwendet werden, wenn Sie die Schritte Tabellenerstellung und Datenladen vollständig voneinander trennen müssen.
-D Verwenden Sie das PostgreSQL "dump"-Format für die Ausgabedaten. Dies kann mit -a, -c und -d kombiniert werden. Es ist viel schneller zu laden als das standardmäßige SQL-Format "insert". Verwenden Sie diese Option für sehr große Datenmengen.
-s <SRID> Erstellt und füllt die Geometrietabellen mit der angegebenen SRID.
-k Behalten Sie den Fall der Identifikatoren (Spalte, Schema und Attribute). Beachten Sie, dass die Attribute in Shapefile alle UPPERCASE sind.
-i Alle Ganzzahlen auf Standard-32-Bit Ganzzahlen umstellen, keine 64-Bit Bigints erzeugen, auch wenn die DBF-Header-Signatur dies zu rechtfertigen scheint.
-I Erstellen Sie einen GiST-Index auf der Geometrie-Spalte.
-w Output WKT-Format, zur Verwendung mit älteren (0.x) Versionen von PostGIS. Beachten Sie, dass dies zu Koordinatendriften führt und M-Werte aus Shapefiles entfernt.
-W <Encoding> Geben Sie die Kodierung der Eingangsdaten (dbf-Datei) an. Bei der Verwendung werden alle Attribute des dbf von der angegebenen Kodierung nach UTF8 konvertiert. Die resultierende SQL-Ausgabe wird einen Befehl SET CLIENT_ENCODING to UTF8 enthalten, so dass das Backend in der Lage sein wird, von UTF8 in die Codierung umzuwandeln, die die Datenbank intern verwenden soll.

Beispiele:
In diesem Beispiel wird die Datei “mytable” direkt in die Datenbank “mydb” durchgeladen. Dabei wird automatisch die Tabelle „mytable“ angelegt und durch die Verwendung des Pipe-Zeichens (|) direkt in die Datenbank durchgeladen:

```
shp2pgsql -s 25832 -W Latin1 -I myshape  mytable | psql -U postgres -d mydb
```

Sehr komfortabel ist in diesem Zusammenhang auch das Anlegen von .bat-Dateien zum Laden von mehreren Shape-Dateien in einem Ordner, z.B.

```
@echo off
shp2pgsql -s 25832 -W Latin1 -I shape1 tabelle1 | psql -U postgres -d mydb
shp2pgsql -s 25832 -W Latin1 -I shape2 tabelle2 | psql -U postgres -d mydb
shp2pgsql -s 25832 -W Latin1 -I shape3 tabelle3 | psql -U postgres -d mydb
pause
```

Soll der File nicht direkt in die Datenbank durchgeladen werden kann auch ein SQL-File erzeugt werden, der später in der Datenbank ausgeführt wird:

```
shp2pgsql -s 25832 -W Latin1 - myshape  mytable > sqlshape.sql
```

pgsql2shp

Dieses Programm dient dem Export von PostgreSQL/PostGIS nach ESRI-Shape
Hierbei können aus SQL-Statements abgesetzt werden.
Syntax:
pgsql2shp [<options>] <database> [<schema>.]<table>
pgsql2shp [<options>] <database> <query>

Folgende Optionen stehen zur Verfügung:

-f <Dateiname> Schreiben Sie die Ausgabe auf einen bestimmten Dateinamen.
-h <host> Der Datenbank-Host, mit dem eine Verbindung hergestellt werden soll.
-p <port> Der Port, mit dem die Verbindung auf dem Datenbankrechner hergestellt werden soll.
-P <password> Das Passwort, das bei der Verbindung zur Datenbank verwendet werden soll.
-u <user> Der Benutzername, der bei der Verbindung zur Datenbank verwendet werden soll.
-g <Geometrie-Spalte> Im Falle von Tabellen mit mehreren Geometrie-Spalten, die Geometrie-Spalte, die beim Schreiben der Shape-Datei verwendet werden soll.
-b Verwenden Sie einen binären Cursor. Dies macht die Operation schneller, funktioniert aber nicht, wenn ein NON-Geometrieattribut in der Tabelle keinen Cast in Text enthält.
-r Raw-Modus. Lassen Sie das Gid-Feld und die Namen der Escape-Spalten nicht fallen.
-d Aus Gründen der Abwärtskompatibilität: Schreiben einer 3-dimensionalen Shape-Datei beim Dumping aus alten (vor 1.0.0.0) Postgis-Datenbanken (die Vorgabe ist, in diesem Fall eine 2-dimensionale Shape-Datei zu schreiben). Ab postgis-1.0.0.0+ sind die Dimensionen vollständig kodiert.

Beispiele:
In diesem Beispiel wird die gesamte Tabelle „mytable“ in den Shapefile shapename geschrieben:

```
pgsql2shp -f C:\meinverzeichnis\shapename mydb -h localhost -P xxxx -u postgres mytable
```

In diesem Beispiel werden über eine Boundingbox alle darin liegenden Objekte der Tabelle „mytable“ ausgewählt:

```
pgsql2shp -f C:\meinverzeichnis\shapename -h localhost -P xxxx -u postgres mydb "SELECT * FROM mytable  WHERE the_geom && SetSRID('BOX3D(2618601 5580443,2634543 5594406)'::box3d,25832);"
```

ogr2ogr

dieses Tool ist ein Kommandozeilentool aus **Gdal**, eine Übersetzer-Bibliothek für Raster- und Vektor-Geodatenformate, die unter einer Open Source-Lizenz von der Open Source Geospatial Foundation veröffentlicht wird. Nähere Informationen finden Sie hier: http://www.gdal.org/
Dieses Programm kann verwendet werden, um zahlreiche Vektorformate zwischen Dateiformaten zu konvertieren, zu transformieren oder anders zu verarbeiten.
Der Funktionsumfang ist sehr mächtig und soll an dieser Stelle nur gestreift werden.

Mit folgendem Befehl importieren Sie einen Shapefile in die Postgis Datenbank:

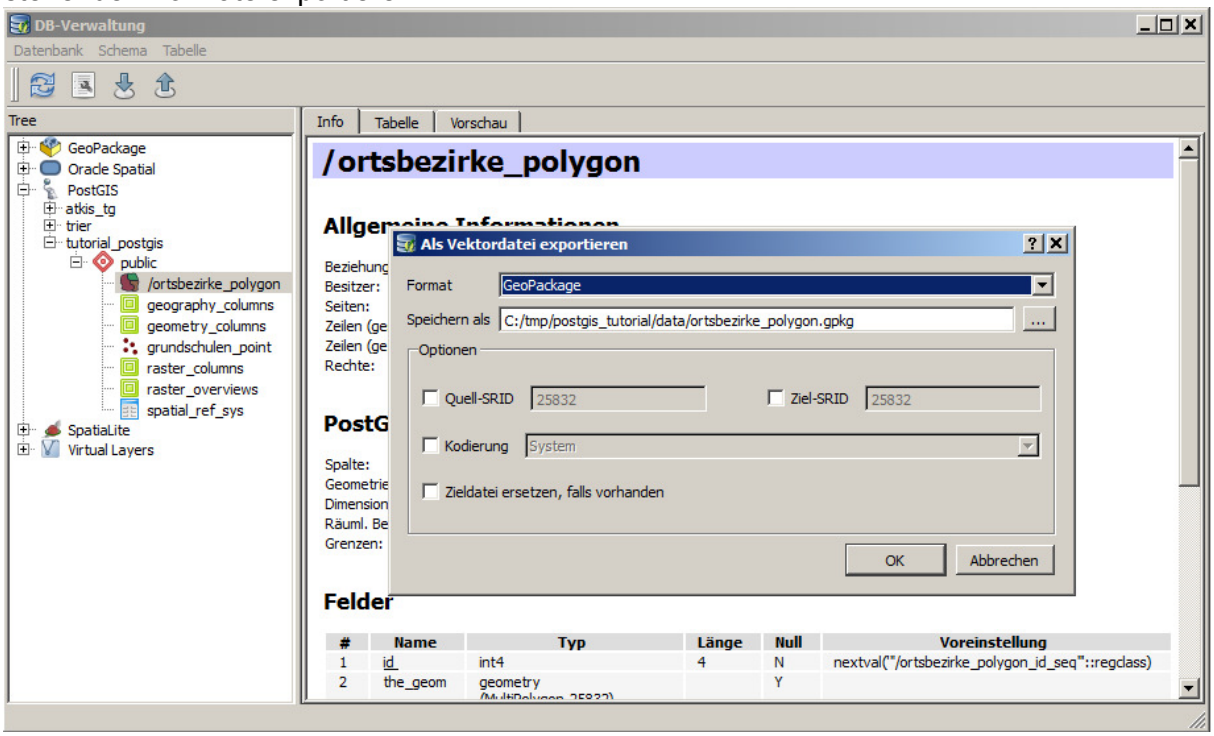
```
ogr2ogr -f "PostgreSQL" PG:"host=localhost user=postgres dbname= dbname password=xxxx" -nlt GEOMETRY C:/tmp/shapefilename.shp
```

Mit folgendem Befehl exportieren Sie eine Postgistabelle in einen Shapefile:

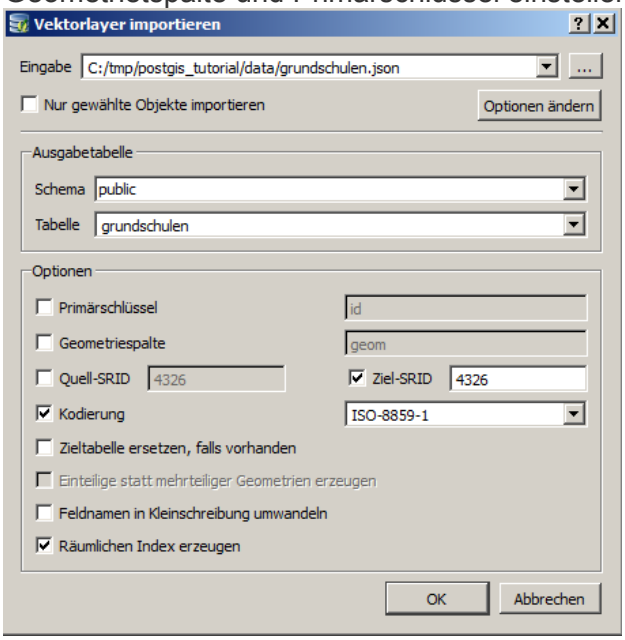
```
ogr2ogr -f "ESRI Shapefile" "C:/tmp/shapefilename.shp" "PG:dbname=dbname host=localhost port=5432 user=postgres password=xxxx tables=tabname" -T_SRS EPSG:25832
```

4.3 Import und Export mit QGIS

QGIS is eines der fortschrittlichsten OpenSource Desktop Gis Programme. Sie können es kostenlos unter https://www.qgis.org herunterladen und unter Windows installieren.
Die Datenaustausch und die Einbindung von Postgis Tabellen ist sehr komfortabel gelöst. Unter dem Menüpunkt Datenbank -> DBVerwaltung gibt es einen Dialog um Daten aus einem QGIS Projekt direkt in Postgis zu importieren oder eine Tabelle zu exportieren.
Im Hintergrund arbeitet auch hier das bereits erwähnte Tool ogr2ogr.
Sie müssen zunächst eine Verbindung zu der Postgis Datenbank aufbauen und abspeichern. Danach können Sie direkt auf die Tabellen zugreifen und z.B. in alle zur Verfügung stehenden Formate exportieren.



Der umgekerte Weg eines Imports funktioniert ähnlich. Sie müssen einen Namen für die Zieltabelle vergeben und ggfls. das Ziel- bzw. Quell Koordinatensystem angeben.
Ebenso können Sie einen räumlichen Index erzeugen lassen und den Namen der Geometrietspalte und Primärschlüssel einstellen.



5. Einbindung in externe Software

Es gibt mittlerweile zahlreiche GIS Software die Postgis bzw. PostgreSQL unterstützen. Neben Projekten aus dem OpenSource Bereich (z.B. MapServer, QGIS, uDig, Geoserver, GRASS GIS und gvSIG) haben auch zahlreiche proprietäre Anbieter entsprechende Schnittstellen geschaffen. An dieser Stelle sollen sellvertretend nur 2 Varianten dargestellt werden.

5.1 UMN Mapserver

Der UMN Mapserver ist ein klassischer Vertreter aus dem WebGIS-Server Bereich. Postgis-Tabellen können direkt in UMN Mapserver eingebunden werden.
Dabei wird im Layerobjekt der Connectentyp auf postgis gesetzt und das data-Objekt mit dem SQL-Statement belegt:

```
LAYER
 NAME "testlayer"
 TYPE polygon
 STATUS on
 CONNECTIONTYPE postgis
 CONNECTION "user=postgres password=xxxx dbname=mydb  host=localhost port=5432"
 DATA "the_geom from mytable  using unique gid"
 CLASS
    NAME "test"
    STYLE
     OUTLINECOLOR 200 0 200
   END
  END
END
```

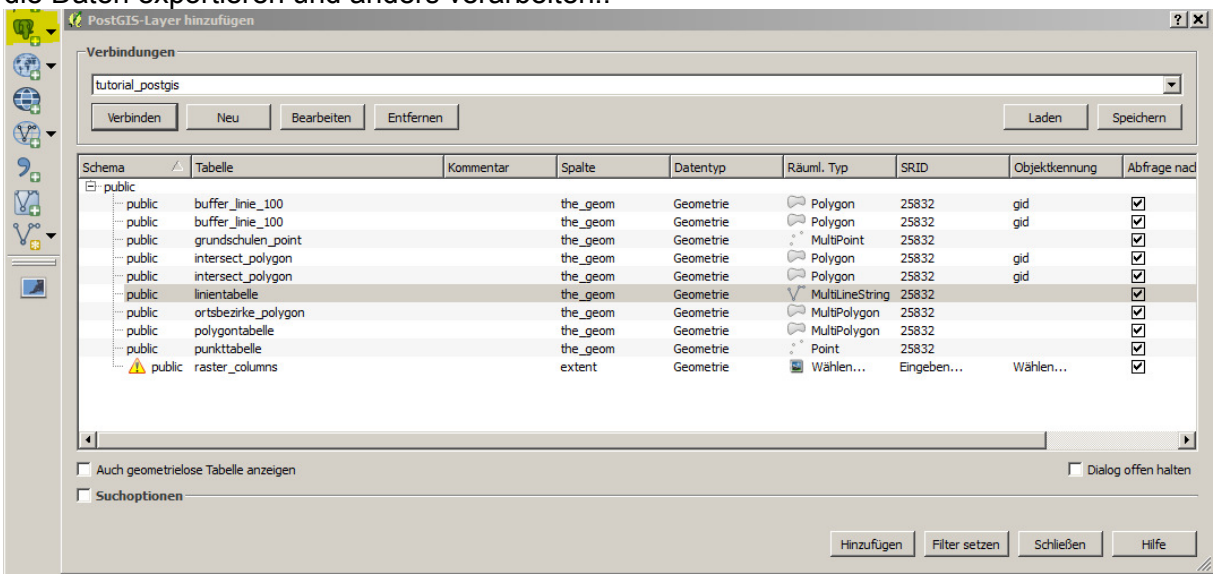
Es Können auch komplexere Statement abgesetzt werden. Im folgenden Beispiel werden die Mittelpunktkoordinaten aus einem Polygontabelle abgefragt und diese als Punktthema dargestellt:

```
LAYER
 NAME "testlayer "
 TYPE point
 STATUS on
 CONNECTIONTYPE postgis
 CONNECTION "user=postgres password=xxxx dbname=mydb host=localhost port=5432"
 DATA "the_geom FROM (SELECT gid, ST_GeomFromText('POINT(' || ST_X(ST_Centroid(the_geom)) || ' ' ||
  ST_Y(ST_Centroid(the_geom)) || ')', 25832) AS the_geom FROM mytable) as foo using unique gid using SRID=25832' "
 TEMPLATE "template/query.html"
  CLASS
   NAME "test"
   SYMBOL "testsymbol"
   SIZE 20
   END
END
```

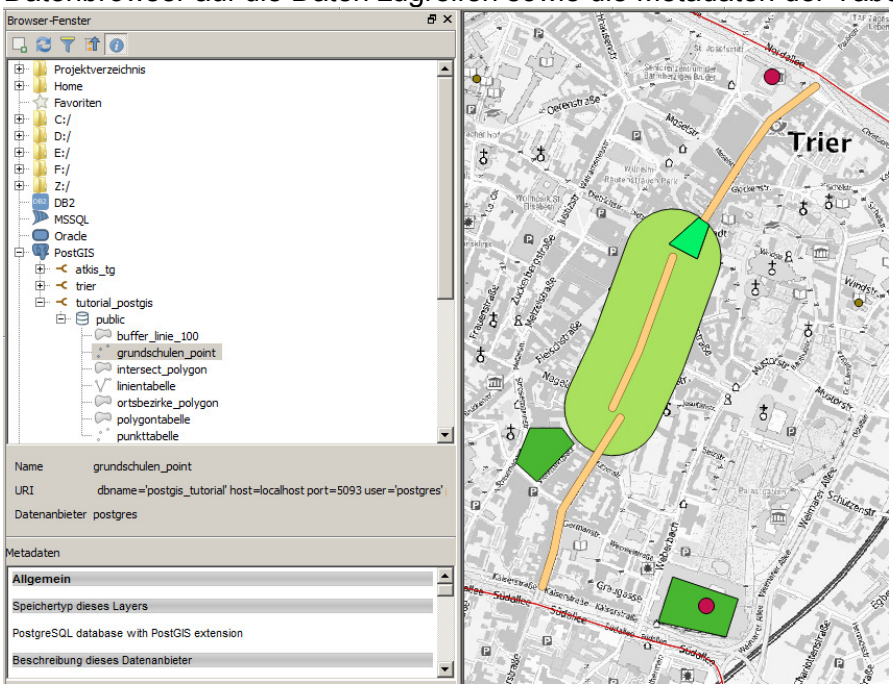
Weitere Informationen finden Sie auf der Homepage von Mapserver:
http://mapserver.org/input/vector/postgis.html

5.2 QGIS

QGIS hat sich mittlerweile als Quasi Standard des OpenSource Desktop GIS Bereichs gemausert. Die Software bietet eine sehr komfortable Einbindung von Postgis Daten über die universelle Datenbank-Schnittstelle. Zudem gibt es mittlerweile zahlreiche Plugins die mit dem Postgis Datenbankserver kommunizieren
Zum Einbinden einer Postgis Datenbank gehen Sie auf das Elefanten Symbol links in der Leiste. Um eine Datenbankverbindung aufzubauen klicken Sie auf Neu und geben die Verbindungsdaten zum Datenbankserver ein. Sie können dabei Ihre Zugansdaten lokal abspeichern.
Danach klicken Sie auf "verbinden" und alle verwendbaren Geometrietabellen erscheinen in der Liste. Wählen Sie das gewünschte Thema aus und fügen es dem Projekt hinzu.
Sofern Sie entsprechende Rechte besitzen können Sie direkt auf dem Thema editieren oder die Daten exportieren und anders verarbeiten..



Sollte eine Tabelle im Dialog nicht hinzufügbar sein müssen Sie eine Objektkennung auswählen oder einen SRID (EPSG Code) eingeben. Dieses Problem kommt häufig bei Sichten (Views) vor. Ist eine Verbindung gespeichert, können Sie einfach über den Datenbrowser auf die Daten zugreifen sowie die Metadaten der Tabellen erkunden.



6. Räumliche Operationen mit Postgis

Basiswissen SQL:
Ein Grundwissen bzgl. SQL (Structured Query Language) als Standardsprache für den Zugriff auf Datenbanken wird an dieser Stelle vorausgesetzt.

6.1 Basiswissen zu PostgreSQL und PostGIS

Groß- und Kleinschreibung:
PostgreSQL interpretiert alle Schlüsselwörter und Bezeichner als Kleinbuchstaben. Deshalb sollten Tabellen- und Feldnamen in Kleinbuchstaben gehalten werden, um Fehler zu vermeiden.

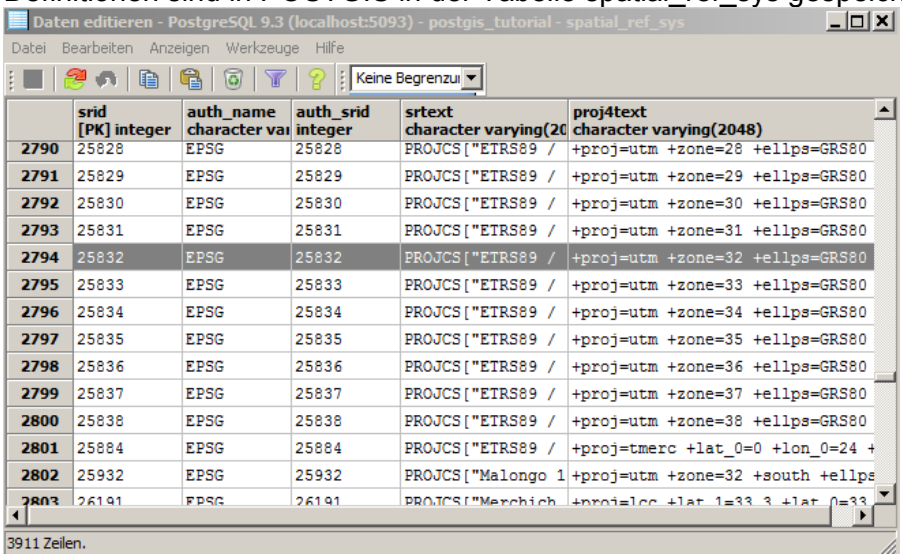
Wichtige Datentypen:
In der Regel kommen nur recht wenige der zahlreichen verfügbaren Datentypen zum Einsatz. Hier eine Auswahl der meist gebräuchlichen Datentypen:

int4, int8 → ganze Zahlen (4byte/8byte)
float8 → Dezimalzahlen (double precision)
varchar → Textfelder mit variabler Zeichenlänge
text → Memofeld, Textfeld mit unbegrenzter Länge
bool → Boolean (true/false)
date → Datumsangebe im Standardformat YYYY-MM-DD, z.B. 2009-10-01

OGC Konform
Die Datenhaltung einer räumlichen Tabelle in PostgreSQL mit PostGIS folgt den *OGC Simple Feature Specifications für SQL*. Die OGC Spezifikation definiert ein SQLSchema zur Verwaltung von Geodaten.
Eine Feature Table enthält in der Regel einen Geometrietyp (Punkt, Linie, Polygon). Ausnahme: Geometrycollection. Die Geometrieobjekte werden dabei in eigenem Tabellenfeld gespeichert. Das Format der Geometrieobjekte ist das WKT-Format (Well Known Text), intern werden die Geometrien im WKB-Format (Well Known Binary) gehalten. Hier eine Übersicht der Geometrietypen im WKT-Format (Quelle: OpenGIS Simple Features Specification for SQL):

Geometry Type	SQL Text Literal Representation	Comment
Point	`'POINT (10 10)'`	a `Point`
LineString	`'LINESTRING ( 10 10, 20 20, 30 40)'`	a `LineString` with 3 points
Polygon	`'POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))'`	a `Polygon` with 1 exterior ring and 0 interior rings
Multipoint	`'MULTIPOINT (10 10, 20 20)'`	a `MultiPoint` with 2 point
MultiLineString	`'MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))'`	a `MultiLineString` with 2 linestrings
MultiPolygon	`'MULTIPOLYGON ( ((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60 ) ))'`	a `MultiPolygon` with 2 polygons
GeomCollection	`'GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))'`	a `GeometryCollection` consisting of 2 `Point` values and a `LineString` value

Koordinatensysteme:
Jeder in Postgis gespeicherte Datensatz (egal ob Vektor- und Rasterdaten) muss einem Koordinatensystem zugeordnet werden. Dafür werden sogenannte EPSG Codes benutzt. Die European Petroleum Survey Group Geodesy (EPSG) war eine Arbeitsgruppe der europäischen Öl- und Gaserkundungsunternehmen, die durch ihr System weltweit eindeutiger Schlüsselnummern geodätischer Datensätze (EPSG-Codes) bekannt wurde. Der EPSG-Code ist ein System weltweit eindeutiger, 4- bis 5-stelliger Schlüsselnummern (SRIDs) für Koordinatenreferenzsysteme und andere geodätische Datensätze. Diese Definitionen sind in POSTGIS in der Tabelle spatial_ref_sys gespeichert:

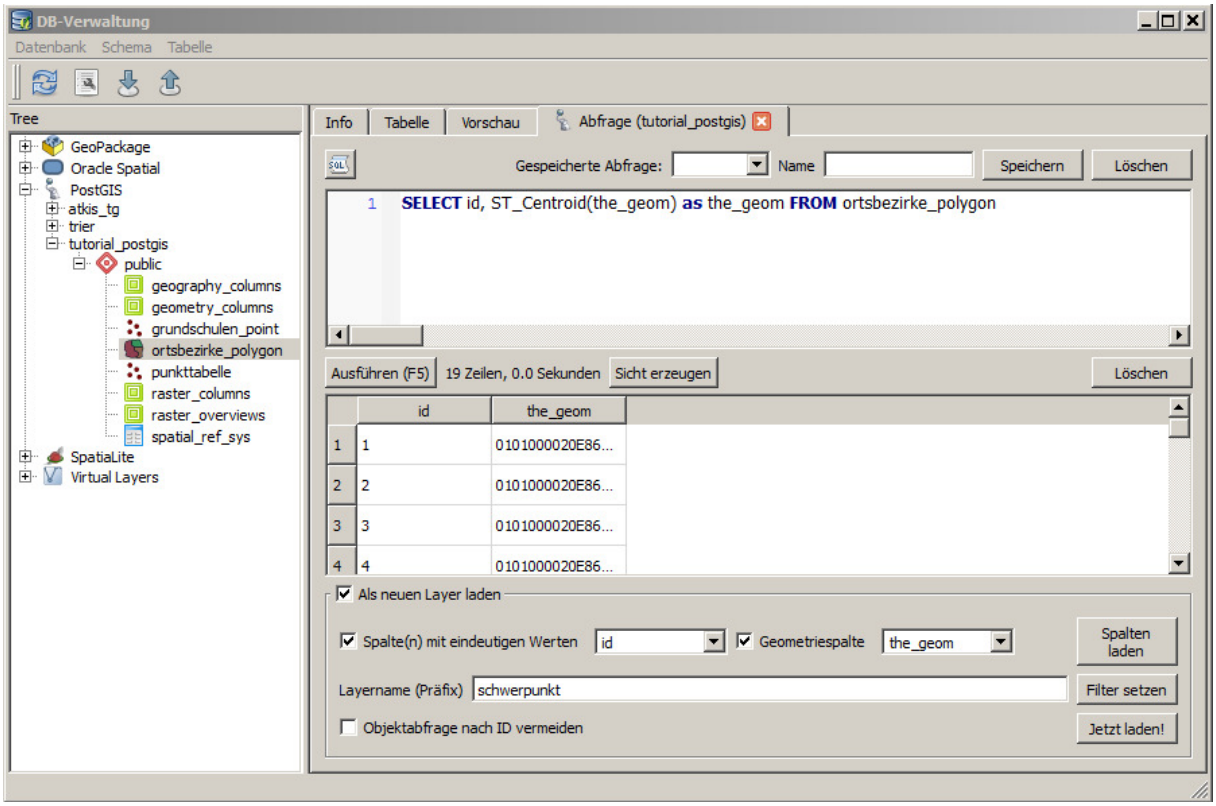


Der EPSG Code für das amtliche Koordinatensystem Rheinland-Pfalz ist z.B. 25832 (ETRS 89 UTM Zone 32). Der wohl am häufigsten verwendete EPSG-Code ist der 4326 für das World Geodetic System 1984 (kurz WGS 84).
Hier eine Liste weiterer in Deutschland gebräuchlichen EPSG Codes und deren Koordinatenreferenzsystem:

Code	Koordinatenreferenzsystem	Bemerkung
4326	WGS-84 / geographisch 2D	weltweites System für GPS-Geräte, OSM Datenbank
25832	ETRS89 / UTM Zone 32N	von 6° O bis 12° O - Deutschland (W+M)+Österreich (W) Schweiz
25833	ETRS89 / UTM Zone 33N	von 12° O bis 18° O - Deutschland (O)+Österreich (M+O)
31466	DHDN / Gauß-Krüger Zone 2	Deutschland - westlich von 7,5° O
31467	DHDN / Gauß-Krüger Zone 3	Deutschland - von 7,5° O bis 10,5° O
31468	DHDN / Gauß-Krüger Zone 4	Deutschland - von 10,5° O bis 13,5° O
31469	DHDN / Gauß-Krüger Zone 5	Deutschland - von 13,5° O bis 16,5° O[
3857	WGS 84 / Pseudo-Mercator	Google Maps, OSM, Bing Maps u.a.

6.2 Erzeugen von Geodaten via SQL

All im folgenden Kapitel aufgeführten SQL Befehle können einfach im SQL Editor von pgAdmin oder direkt auf der Konsole ausgeführt werden.
Eine weitere Möglichkeit bietet die DB-Verwaltung in QGIS.
Der Vorteil hier ist, daß Sie immer direkt eine grafische Kontrolle bekommen und sogar direkt neue Ebenen in Ihrem QGIS Projekt erzeugen können.
Dazu öffnen Sie im Menü Datenbank -> DB-Verwaltung den Dialog und erzeugen eine entsprechende Verbindung zu Ihrer Datenbank.
Danach können Sie über den Reiter "Abfrage" direkt SQL Statements eingeben und ausführen. Für eine Postgis Tabelle benötigen Sie mindestens eine Spalte für den Primärschlüssel, also eine Spalte mit eindeutigen Werten, und eine Spalte für die Geometrie.
Danach können Sie aus jedem gültigen SQL Befahl eine neue Ebene erzeugen oder diesen Befehl direkt auf der Datenbank in einer sogenannten Sicht (View) abspeichern.



Beispiel Anlegen eines neuen Punktthemas mit SQL:

→ Anlegen einer Tabelle, wir erzeugen eine Id (gid) mit dem Datentyp serial. Damit wird bei jedem Einfügen automatisch immer eine Ganzzahl hochgezählt und so ein eindeutiger Wert für die ID bzw. den Primärschlüssel erzeugt. Zusätzlich wird das Feld "titel" als Freitextfeld eingefügt.

```
CREATE TABLE punkttabelle (gid serial, titel  varchar);
```

→ Einfügen der Geometriespalte und Zuordnung des Koordinatensystems über die Funktion **AddGeometryColumn()**. Dabei wird neben dem Schema, der Zieltabelle und dem Namen der Spalte auch auch der EPSG Code und die Dimension eingegeben.

```
SELECT AddGeometryColumn('public','punkttabelle','the_geom',25832,'POINT',2);
```

→ Bei der Sichtung der Tabelle im pgadmin kommt der berechtigte Hinweis, dass kein Primärschlüssel vorhanden ist, deshalb hier noch das entsprechende SQL:

```
ALTER TABLE punkttabelle ADD CONSTRAINT punkttabelle_pkey PRIMARY KEY(gid);
```

→ Gerade bei großen Tabellen sollte aus Performancegründen ein räumlicher Index auf die Geometriespalte gelegt werden.

```
CREATE INDEX punkttabelle_the_geom_gist ON punkttabelle  USING gist (the_geom);
```

→ Einfügen von Daten mit Hilfe der Funktion **ST_GeomFromText()**
Die Geometriedaten werden hier in Form des WKT-Formats eingefügt.

```
INSERT INTO punkttabelle (titel,the_geom) values ('Porta Nigra', ST_GeomFromText ('POINT(330278 5514589)', 25832));
INSERT INTO punkttabelle (titel,the_geom) values ('Kaiserthermen', ST_GeomFromText ('POINT(330140 5513488)',
25832));
INSERT INTO punkttabelle (titel,the_geom) values ('Römerbrücke', ST_GeomFromText ('POINT(329037 5513750)',
25832));
```

Die Daten sind jetzt eingefügt und können bereits mit einem Client dargestellt werden.
Hier noch einmal die komplette Syntax zum Erzeugen einer räumlichen Punkttabelle mit Bordmittel von POSTGIS:

```
CREATE TABLE punkttabelle (gid serial, titel  varchar);
SELECT AddGeometryColumn('public','punkttabelle','the_geom',25832,'POINT',2);
ALTER TABLE punkttabelle ADD CONSTRAINT punkttabelle_pkey PRIMARY KEY(gid);
CREATE INDEX punkttabelle_the_geom_gist ON punkttabelle  USING gist (the_geom);
INSERT INTO punkttabelle (titel,the_geom) values ('Porta Nigra', ST_GeomFromText ('POINT(330278 5514589)', 25832));
INSERT INTO punkttabelle (titel,the_geom) values ('Kaiserthermen', ST_GeomFromText ('POINT(330140 5513488)',
25832));
INSERT INTO punkttabelle (titel,the_geom) values ('Römerbrücke', ST_GeomFromText ('POINT(329037 5513750)',
25832));
```

Das Erzeugen von Linien bzw. Polygonthemen funktioniert analog unter Verwendung der entsprechenden WKT Syntax, z.B.

```
LINESTRING(0 0,1 1,1 2)
POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
```

Besser ist die direkte Verwendung von Multiobjekttypen

```
MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
```

Hier die Syntax für die Erzeugung einer Linientabelle und einer Polygontabelle und dem Insert von 3 Testobjekten zu späteren Übungszwecken:

```
CREATE TABLE linientabelle (gid serial, titel  varchar);
SELECT AddGeometryColumn('public','linientabelle','the_geom',25832,'MULTILINESTRING',2);
ALTER TABLE linientabelle ADD CONSTRAINT linientabelle_pkey PRIMARY KEY(gid);
CREATE INDEX linientabelle_the_geom_gist ON linientabelle  USING gist (the_geom);
INSERT INTO linientabelle (titel,the_geom) values ('Fleischstraße', ST_GeomFromText ('MULTILINESTRING((330057
5514238,329851 5514048,329754 5513970))', 25832));
INSERT INTO linientabelle (titel,the_geom) values ('Simeonstraße', ST_GeomFromText ('MULTILINESTRING((330137
5514292,330222 5514426,330275 5514499,330368 5514569))', 25832));
INSERT INTO linientabelle (titel,the_geom) values ('Neustraße', ST_GeomFromText ('MULTILINESTRING((329800
5513528,329824 5513602,329841 5513687,329962 5513879))', 25832));

CREATE TABLE polygontabelle (gid serial, titel  varchar);
SELECT AddGeometryColumn('public','polygontabelle','the_geom',25832,'MULTIPOLYGON',2);
ALTER TABLE polygontabelle ADD CONSTRAINT polygontabelle_pkey PRIMARY KEY(gid);
CREATE INDEX polygontabelle_the_geom_gist ON polygontabelle  USING gist (the_geom);
```

```
INSERT INTO polygontabelle (titel,the_geom) values ('Kaiserthermen', ST_GeomFromText ('MULTIPOLYGON(((330182
5513423,330041 5513454,330071 5513548,330207 5513502,330182 5513423)))', 25832));
INSERT INTO polygontabelle (titel,the_geom) values ('Hauptmarkt', ST_GeomFromText ('MULTIPOLYGON(((330147
5514281,330122 5514210,330063 5514241,330125 5514296,330147 5514281)))', 25832));
INSERT INTO polygontabelle (titel,the_geom) values ('Viehmarkt', ST_GeomFromText ('MULTIPOLYGON(((329866
5513825,329789 5513746,329745 5513786,329774 5513858,329841 5513858,329866 5513825)))', 25832));
```

6.3 Häufige SQL Funktionen in Postgis

Postgrsql- / Postgis Version ermitteln:

```
select version() as version_postgresql, postgis_full_version() as version_postgis;
```

Fehler in Geometrien aufspüren:

→ Mit der Funktion **ST_IsValid()** können Sie überprüfen, ob eine Geometrie “Wohlgeformt” ist also eine OGC-Konforme Geometrie ist:

```
SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As konform,
        ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As defekt
```

Ausgabe von Geometriedaten im WKT-Format:

→ Um Daten einer bestehenden Tabelle wieder in einem lesbaren WKT-Format auszugeben dient die Funktion **ST_AsEWKT()**:

```
Select ST_AsEWKT(the_geom) from punkttabelle;
```

Ausgegeben wird der EPSG-Code in Verbindung mit dem WKT-String:

```
> SRID=25832;POINT(330278 5514589)
> SRID=25832;POINT(330140 5513488)
> SRID=25832;POINT(329037 5513750)
```

Ausgabe der räumlichen Ausdehnung:

→ Die räumliche Ausdehnung lässt sich mit der Funktion **ST_EXTENT()** ausgeben. Dabei wird eine Bounding-Box der begrenzenden Koordinaten zurückgegeben (südwestlichster Punkt / nordöstlichster Punkt):

```
SELECT ST_EXTENT(the_geom) FROM punkttabelle;
```

Ausgabe:

```
> BOX(329037 5513488,330278 5514589)
```

Anlegen einer weiteren Geometriespalte und Transformation nach WGS84

→ man kann beliebig viele Geometriespalten anlegen. Mit der Funktion **ST_TRANSFORM()** können die Geometrien einfach in ein anderes Koordinatensystem transformiert werden

```
ALTER TABLE punkttabelle ADD COLUMN the_geom_wgs geometry(Point,4326);
update punkttabelle set the_geom_wgs = ST_Transform(the_geom, 4326);
```

Ausgabe von Punktkoordinaten:

→ Um an die X- bzw Y Koordinaten (double) eines Punktthemas direkt auszugeben kommt die Funktion **ST_X()** bzw. **ST_Y()** zum Einsatz:

```
SELECT ST_X(the_geom), ST_Y(the_geom) FROM punkttabelle;
```

→ Um z.B. an die Anzahl der Stützpunkte eines Linienobjektes zu erfahren, können Sie die Funktion **ST ST_NPoints** einsetzen:

```
SELECT ST_NPoints(the_geom) FROM linientabelle;
```

→ Um den Centroid eines Polygons, Multipoints oder Linestrings auszugeben verwenden Sie **ST_Centroid()** (in diesem Beispiel in Verbindung mit der Ausgabe als WKT):

```
SELECT ST_AsText(ST_Centroid(the_geom)) FROM polygontabelle;
```

→ Um zu erzwingen das der Mittelpunkt eines Polygons auf der dazugehörigen Fläche liegt, verwenden Sie **ST_PointOnSurface ()**. In diesem Beispiel wird die id-Spalte mit der neu erzeugten Geometriespalte einer Polygontabelle abgefragt.

```
SELECT gid, ST_PointOnSurface(the_geom) as the_geom FROM polygontabelle;
```

Ausgabe von Flächen und Längen sowie Umfang:

→ Die Funktion **ST_AREA()** gibt die Fläche eines Polygons als double-Wert zurück (an dieser Stelle ein kleiner Ausflug, **ilike** hilft Ihnen bei PostgreSQL einen Textstring zu finden ohne auf Groß oder Kleinschreibung zu achten):

```
SELECT ST_AREA(the_geom) FROM polygontabelle WHERE titel ilike 'viehmarkt';
```

→ Die Funktion **ST_LENGTH()** gibt die jeweilige Länge eines Linienobjektes zurück

```
SELECT ST_LENGTH(the_geom) as laenge_m FROM linientabelle;
```

→ Die Funktion **ST_PERIMETER()** gibt den jeweiligen Umfang eines Polygonobjektes zurück

```
SELECT ST_PERIMETER(the_geom) FROM polygontabelle;
```

6.4 Geometrische Operationen

→ Einen Buffer von z.B. 100 Metern auf alle Objekte eines Linienthemas erzeugen wir mit der Funktion **st_buffer():**

```
SELECT st_buffer(the_geom,100) as buffer_100 FROM linientabelle;
```

→ Um diese Thema auch in einem Client (z.B. QGIS) als hinzufügbares Thema verwenden zu können und nur auf ein Objekt zu beschränken (Name1), bilden wir einen View auf das Bufferthema. Die Funktion **ST_SetSRID** dient in diesem Zusammenhang einer Zuordnung zu einem EPSG-Code.

```
CREATE VIEW buffer_linie_100 AS
 SELECT gid, titel, ST_SetSRID(st_buffer(the_geom,100), 25832) AS
 the_geom  FROM linientabelle WHERE titel = 'Fleischstraße';
```

→ Die Funktion **ST_Intersection** ermöglicht klassische Verschneidungsoperationen in PostGIS, zurück bekommt man die Verschneidungsgeometrie für die Objekte die nicht leer sind (**ST_IsEmpty** = FALSE):

```
SELECT p.gid, p.titel, ST_SetSRID(ST_Intersection(p.the_geom, b.the_geom) ,25832)
 AS the_geom  FROM polygontabelle AS p, buffer_linie_100 AS b
WHERE ST_IsEmpty (ST_Intersection(p.the_geom, b.the_geom) ) = FALSE;
```

→ Um dieses Thema wiederum in einem Client sichtbar zu machen erzeugen wir einen View:

```
CREATE VIEW intersect_polygon (gid,name,the_geom) AS
SELECT p.gid, p.name, ST_SetSRID(ST_Intersection(p.the_geom, b.the_geom) ,25832)
 AS the_geom  FROM polygontabelle AS p, buffer_linie_100 AS b
WHERE IsEmpty (ST_Intersection(p.the_geom, b.the_geom) ) = FALSE;
```

→ Um eine konvexe Hülle um eine Punktwolke zu erzeugen benötigen wir die Funktion **ST_ConvexHull** und **ST_Union**

```
SELECT 1 as gid, ST_SetSRID(ST_ConvexHull(ST_Union(the_geom )),25832) as the_geom FROM punkttabelle;
```

→ Um festzustellen ob ein Polygon einen Punkt enthält, kann die Funktion **ST_Contains** zur Anwendung kommen:

```
SELECT po.titel AS Polygon_name, pu.titel AS Punkt_name
 FROM polygontabelle AS po, punkttabelle AS pu
 WHERE ST_Contains (po.the_geom, pu.the_geom ) = TRUE;
```

→ Ein weiterer Lösungsweg ist die Funktion **ST_Within**, hier wird allerdings ermittelt, ob die Geometrien vollständig in den Polygonen enthalten sind, dies macht natürlich nur mit Linien und Polygonen Sinn, da Punkte nicht nur teilweise in einem Polygon enthalten sein können:

```
SELECT po.titel AS Polygon_name, pu.titel AS Linien_name
 FROM buffer_linie_100 AS po, linientabelle AS pu
 WHERE ST_Within (pu.the_geom, po.the_geom ) = TRUE;
```

6.5 Weitere Beispiele aus der Praxis

Attribute eines aus einem Thema (usergrenzen) in ein Feld eines anderen Themas (biogas.edituser) übernehmen, wenn Objekt darin liegt:

```
UPDATE biogas SET edituser = u.edituser FROM usergrenzen AS u WHERE ST_CONTAINS(u.the_geom, biogas.the_geom);
```

Linienobjekt aus WKT einfügen:

```
INSERT INTO "messprofile"
("projekt","profil","datum","edituser",the_geom) VALUES
('Baldeneysee-Elodea','Profil01','01.09.2006','edit', ST_GeomFromEWKT ('SRID=31467;MULTILINESTRING((3366132.18
5696901.85,3365966.09 5696925.09))'));
```

Geometrietyp ermitteln:

```
select GeometryType(the_geom) from siedlungsflaechen_fnp;
```

Stringverknüpfung herstellen:

```
UPDATE grenzen_gemeinden SET gem_schl= '07' || gemeinde_n;
```

Substrings ermitteln und verknüpfen

```
update gem_fl set gem_schl_neu = SUBSTR(gem_schl,0,6)||'01'||SUBSTR(gem_schl,6,4);
```

Feld hinzufügen und mit festem Wert besetzen:

```
ALTER TABLE immobilien ADD COLUMN edituser character varying(50);
update immobilien set edituser = 'administrator';
```

Feld in zu einen anderen Typ casten:

```
update gemarkungen set gem_nr_new = cast(gem_nr as integer);
```

Punktview aus zwei Koordinaten einer nichträumlichen Tabelle erzeugen:

```
CREATE VIEW geopoint AS
SELECT lfd_nr, map_text,
ST_SetSRID(ST_GeomFromText ('POINT(' || r_gk2 ||' '|| h_gk2 || ')', 25832), 25832 ) AS the_geom
FROM kultur;

CREATE VIEW geopoint_wgs AS
SELECT lfd_nr, map_text,
ST_SetSRID(ST_GeomFromText ('POINT(' || x_wgs84 ||' '|| y_wgs84 || ')', 4326), 4326 ) AS the_geom
FROM kultur;
```

Ersetzen von Kommas durch Punkte in String und gleichzeitiges Casten:

```
update kultur set x_wgs84 = cast(REPLACE(r_wgs84,',','.') as double precision);
```

Teilabschnitt eines Lineobjekts ausgeben (ST_LINE_Substring), vorher muss im Beispiel diese Geometrie mit ST_GeometryN von einem multiline-Objekt in ein einfaches line-Objekt konvertiert werden:

```
SELECT ST_AsText(ST_Line_Substring(ST_GeometryN(the_geom,1), 0.333, 0.666)) FROM nk_polyline WHERE skey =
'5221042 5319014';
```

Kopieren von ausgewählten Daten in eine andere Tabelle:

```
INSERT INTO baumpflanzungen_2015 (pflanzjahr,the_geom,baumnummer,ga_lang,bemerkung)
SELECT pflanzjahr,the_geom,baumnummer,ga_lang,bemerkung FROM baumpflanzungen_2016;
```

Vergleichen von 2 Spalten:

```
SELECT t1.gdeschl, t1.gdeschl_orig FROM grenzen t1
WHERE
not exists
(SELECT t2.gdeschl_orig FROM grenzen t2
where t2.gdeschl_orig=t1.gdeschl)
```

6.4 Automatisierung durch Triggerfunktionen

Ein Trigger (englisch Auslöser), ist eine Funktion in PostgreSQL oder anderen relationalen Datenbanken. Dabei wird durch eine bestimmte Art der Änderungen von Daten in einer Tabelle (z. B. INSERT, UPDATE, DELETE in SQL) ein gespeichertes Programm aufgerufen, das diese Änderung erlaubt, verhindert und/oder weitere Tätigkeiten vornimmt. An dieser Stelle sollen ein paar Aufgaben mittels Trigger im Zusammenhang mit PostgreSQL und Postgis dargestellt werden.
Zur Erstellung eines Triggers benötigt man zunächst eine Triggerfunktion. Diese Funktion wird dann über den mit der Tabelle verknüpften Trigger aufgerufen.

Zunächst eine Triggerfunktion in der für PostgreSQL üblichen prozedualen Programmiersprache plpgsql, die in einer Punkt-Tabelle "redline_point" nach jedem Update und Insert die Koordinaten Komma separiert in eine Spalte (coord_xy) schreiben soll:

```
CREATE OR REPLACE FUNCTION public.set_coord_xy()
  RETURNS trigger AS
$BODY$
            BEGIN
              NEW.coord_xy = round(st_x(NEW.the_geom)) || ',' || round(st_y(NEW.the_geom));
              RETURN NEW;
            END;
            $BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Der auslösende Trigger auf der Tabelle (redline_point) wird folgendermaßen definiert:

```
CREATE TRIGGER update_coord_xy_redline_point
  BEFORE INSERT OR UPDATE
  ON public.redline_point
  FOR EACH ROW
  EXECUTE PROCEDURE public.set_coord_xy();
```

In einem weiteren fortgeschrittenen Beispiel soll nach jedem Update und Insert der Punkttabelle "redline_point" in einer weiteren Polygon-Tabelle " redline_point_buffer" einen Puffer erzeugt bzw. verändert werden. Dabei wird der Radius und ein paar weitere Spalten aus der Punkttabelle übernommen.
Über entsprechende IF - ELSEIF Schleifen kann auf die entsprechenden SQL Statements reagiert werden.

```
CREATE OR REPLACE FUNCTION public.update_redline_point_buffers()
  RETURNS trigger AS
$BODY$
   BEGIN
      -- INSERT
      IF (TG_OP = 'INSERT') THEN
         INSERT INTO
            public.redline_point_buffer (parent_gid, parent_bufferradius, parent_buffercolor, parent_bufferfillcolor,
edituser, the_geom)
         SELECT
           NEW.gid AS parent_gid,
            NEW.bufferradius AS parent_bufferradius,
            NEW.buffercolor AS parent_buffercolor,
                                        NEW.bufferfillcolor AS parent_bufferfillcolor,
            NEW.edituser AS edituser,
            ST_Buffer(NEW.the_geom, NEW.bufferradius) AS the_geom;
         RETURN NEW;
```

```
    -- UPDATE
    ELSIF (TG_OP = 'UPDATE') THEN
            --Objekt löschen wenn Wert 0
            IF (NEW.bufferradius = 0) THEN
                    UPDATE
                            public.redline_point_buffer AS nrb
                      SET
                            the_geom = NULL,
                            parent_bufferradius = NEW.bufferradius,
                            parent_buffercolor = NEW.buffercolor,
                            parent_bufferfillcolor = NEW.bufferfillcolor
                      WHERE
                            nrb.parent_gid = NEW.gid;
                    RETURN NEW;
       END IF;
      --Buffer aktualisieren wenn Radius größer 0
      IF (NEW.bufferradius > 0) THEN
                      UPDATE
                            public.redline_point_buffer AS nrb
                      SET
                            the_geom = ST_Buffer(NEW.the_geom, NEW.bufferradius),
                            parent_bufferradius = NEW.bufferradius,
                            parent_buffercolor = NEW.buffercolor,
                            parent_bufferfillcolor = NEW.bufferfillcolor
                      WHERE
                            nrb.parent_gid = NEW.gid;
                      RETURN NEW;
      END IF;
      --nix machen wenn null
      IF (NEW.bufferradius IS NULL) THEN
                      RETURN NEW;
      END IF;
    -- DELETE
    ELSIF (TG_OP = 'DELETE') THEN
      DELETE FROM public.redline_point_buffer AS nrb where parent_gid = OLD.gid;
      RETURN OLD;
    END IF;
  END;
$BODY$
 LANGUAGE plpgsql VOLATILE
 COST 100;
```

Der auslösende Trigger auf der Tabelle (redline_point) wird hier folgendermaßen definiert:

```
CREATE TRIGGER trg_update_redline_point_buffers
 AFTER INSERT OR UPDATE OR DELETE
 ON public.redline_point
 FOR EACH ROW
 EXECUTE PROCEDURE public.update_redline_point_buffers();
```

Hier noch eine kleine Auswahl weiterer GIS-Aufgaben die durch Triggerfunktionen erledigt werden können:

Den Flächeninhalt eines Polygons berechnen und in eine Spalte schreiben:

```
CREATE OR REPLACE FUNCTION public.set_area()
  RETURNS trigger AS
$BODY$
            BEGIN
              NEW.area_qm = st_area(NEW.the_geom);
              RETURN NEW;
            END;
            $BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Die Länge einer Polylinie ermitteln und in eine Spalte schreiben:

```
CREATE OR REPLACE FUNCTION public.set_length()
  RETURNS trigger AS
$BODY$
            BEGIN
              NEW.length_m = st_length(NEW.the_geom);
              RETURN NEW;
            END;
            $BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Den Umfang eines Polygons in eine Spalte schreiben:

```
CREATE OR REPLACE FUNCTION public.set_perimeter()
  RETURNS trigger AS
$BODY$
            BEGIN
              NEW.perimeter_m = st_perimeter(NEW.the_geom);
              RETURN NEW;
            END;
            $BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Die X und Y Koordinaten eines Punktthemas in jeweils eine Spalte schreiben:

```
CREATE OR REPLACE FUNCTION public.set_xy()
  RETURNS trigger AS
$BODY$
           BEGIN
             NEW.x_utm = st_x(NEW.the_geom);
             NEW.y_utm = st_y(NEW.the_geom);
             RETURN NEW;
           END;
           $BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```