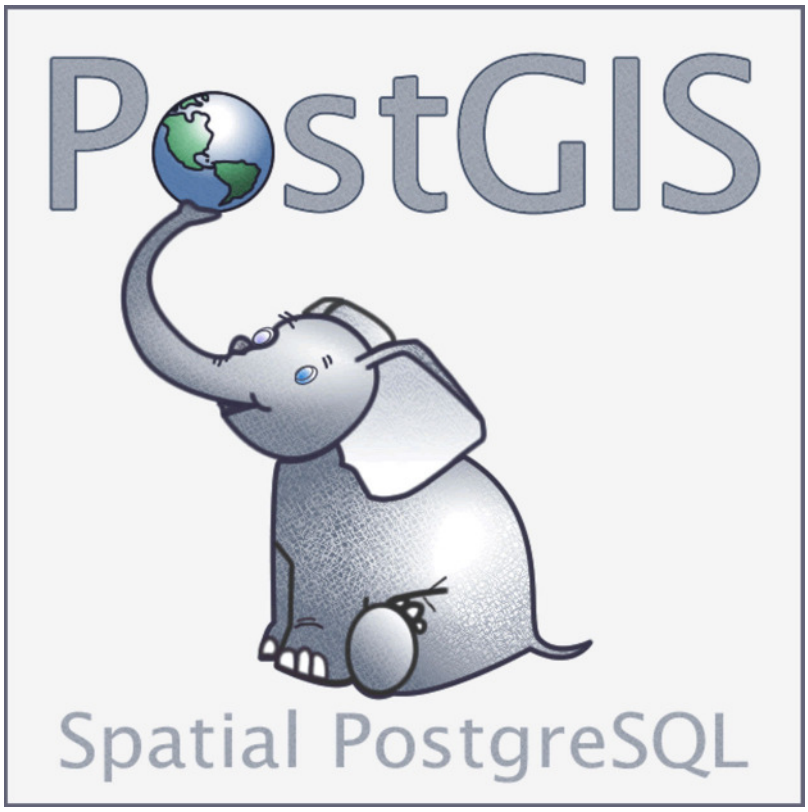
HafenCity Universität Hamburg

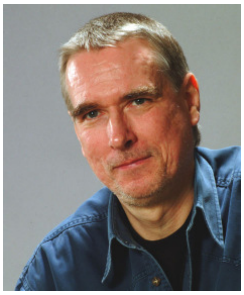
PostGIS 1.5 Tutorial (Grundlagen)



Version 10-06-01

Uwe Dalluege
HCU Hamburg

Autor:
Uwe Dalluege
HafenCity Universität Hamburg
Hebebrandstr. 1
22297 Hamburg
E-Mail: uwe.dalluege@hcu-hamburg.de



Inhaltsverzeichnis

1 Bevor die Post abgeht

PostGIS erweitert das objektrelationale Datenbankmanagementsystem *PostgreSQL* um GIS-Funktionalitäten, die der *OpenGIS* Spezifikation „*OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 2: SQL option*“ folgen
(siehe OGC, http://www.opengeospatial.org/standards/sfs). *PostGIS* wird von der kanadischen Firma *Refractions Research* entwickelt und unterliegt der *GNU General Public License*. Es gibt eine große Funktionsbibliothek, mit denen man z.B. räumliche Abfragen und Analysen durchführen oder Geometrie-Objekte bearbeiten und manipulieren kann.

Die besonderen Merkmale von *PostGIS* sind:

- Basiert auf *OpenGIS* Standards (http://www.opengeospatial.org/standards).
- Unterliegt der *GNU General Public Licence* http://www.gnu.org/copyleft/gpl.html
- Große Funktionsbibliothek zur Manipulation und Analyse geografischer Objekte.
- Wird von vielen GIS-Anwendungen unterstützt.

Um die Funktionalität von *PostGIS* besser demonstrieren zu können, werden hier das Tool *pgAdmin III* und das Programm *OpenJUMP* verwendet. Mit *pgAdmin III* kann man unter anderem eine Verbindung zum Datenbankserver aufbauen, Datenbanken verwalten, Tabellen darstellen oder auch SQL-Anweisungen ausführen.

OpenJUMP ist ein Geoinformationssystem, mit dem man unter anderem auch *PostGIS*-Tabellen darstellen und speichern kann. Eine Beschreibung zu *OpenJUMP* und *PostGIS*-Anbindung finden Sie in dem ***OpenJUMP 1.2 Tutorial (Grundlagen)*** in Kapitel 11.

Hinweis: In *PostGIS* wurde damit begonnen, die Funktionen nach dem *SQL/MM*-Standard zu benennen und sie mit dem Prefix ***ST*** (Spatial Type**)** zu versehen. Die alten Funktionsbezeichnungen bleiben vorübergehend noch bestehen, sollten aber nicht weiter verwendet werden!
In diesem Tutorial werden nur einige wenige *PostGIS*-Funktionen vorgestellt. Eine ausführliche Beschreibung aller Funktionen finden Sie im *PostGIS*-Manual: http://postgis.refractions.net/documentation/

Dieses Tutorial setzt Grundkenntnisse in *SQL* und *OpenJUMP* voraus!

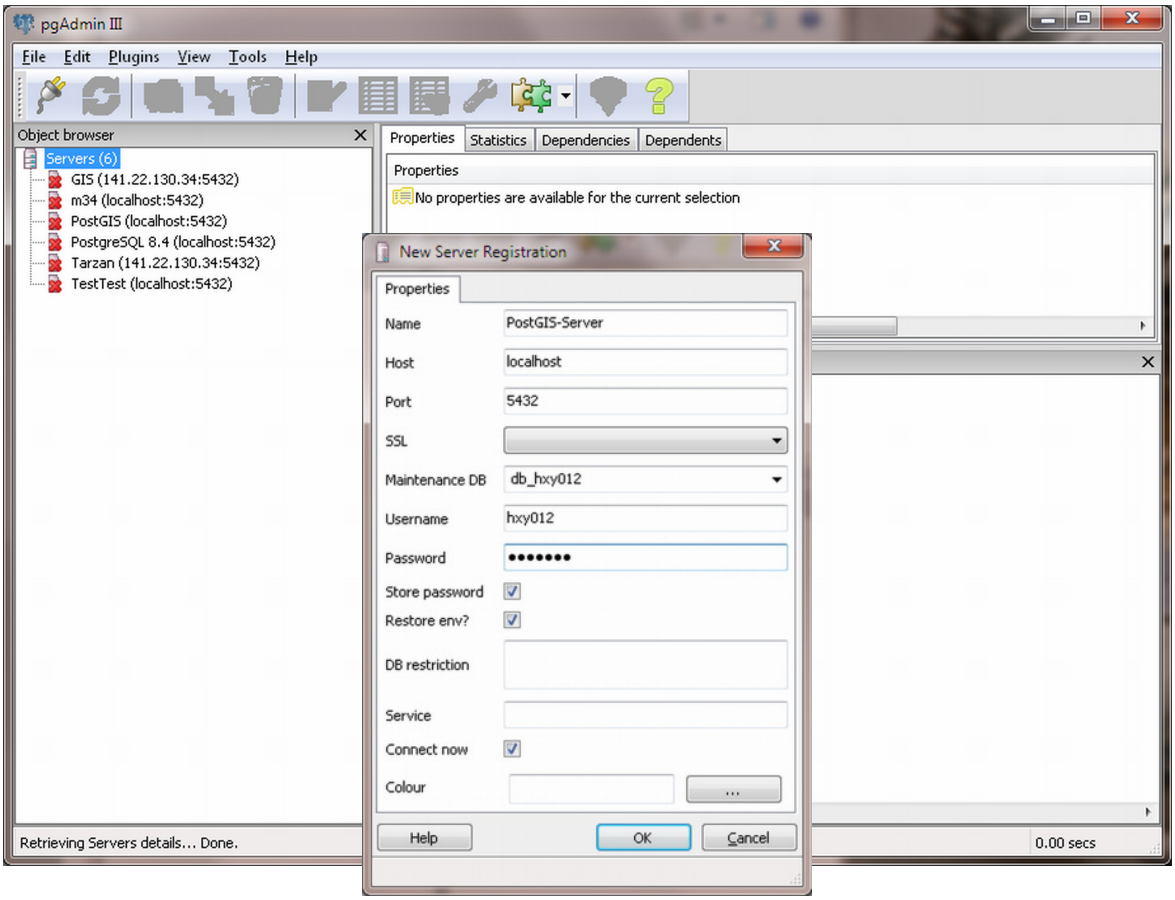
1.1 Die pgAdmin III Oberfläche

1.1.1 Verbindung zum Datenbankserver aufbauen

Bevor wir mit *PostGIS* arbeiten können, müssen wir eine Verbindung zum Datenbankserver (kurz Server) aufbauen. Dazu müssen folgende Informationen bekannt sein:

- Die IP-Adresse des Datenbankservers (**Host;** hier *localhost*).
- Die Portnummer, unter der PostgreSQL angesprochen wird (***Port***; hier *5432*).
- Der Datenbankname (***Maintenance DB***; hier *db_hxy012*).
- Der Benutzername (***Username***; hier *hxy012*).
- Das Benutzerpasswort (***Password***).

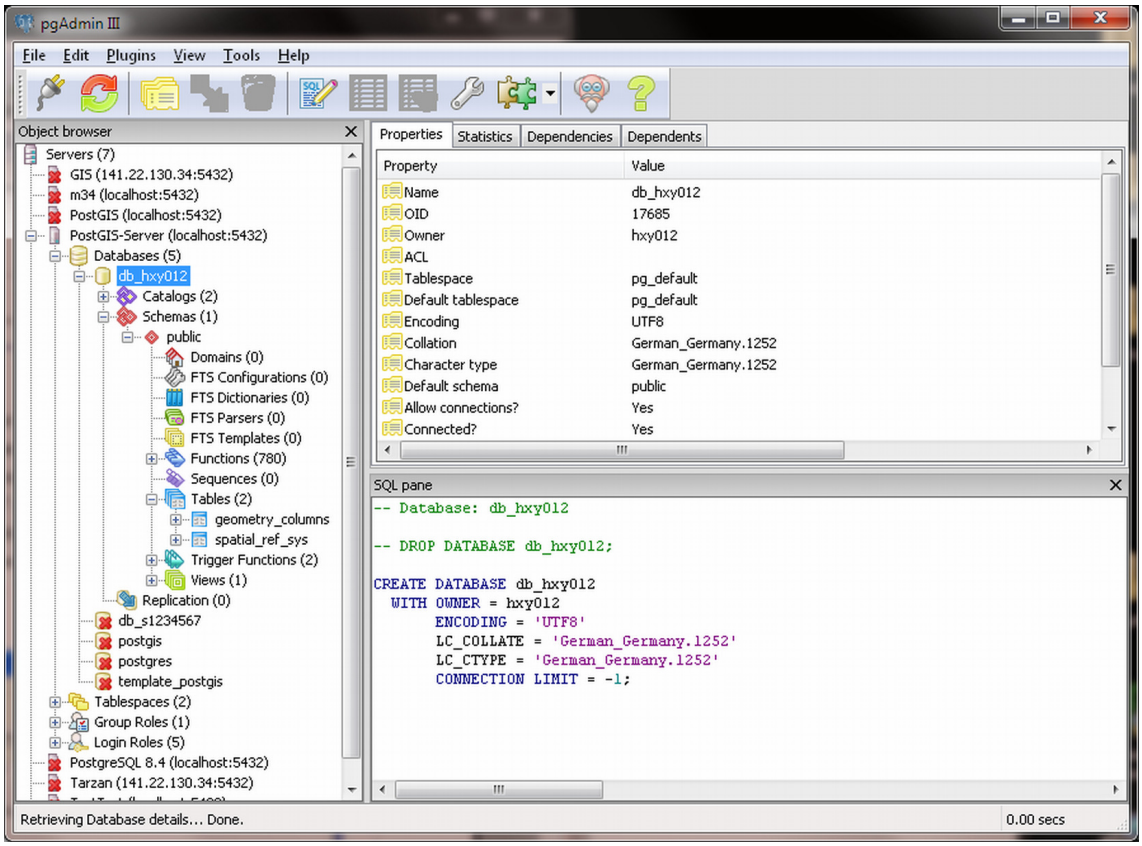
Nach dem Start von *pgAdmin III* klicken Sie auf den Stecker-Knopf (***Add a connection to a server***) und stellen eine Verbindung zum Server her.



Eine Serververbindung hinzufügen.

1.1.2 Die Baumstruktur

Im linken Fenster des *pgAdmin* Tools wird die Baumstruktur der Datenbankserver dargestellt. In der unteren Abbildung wurde nur die Verbindung zu **einem** Server aufgebaut (*PostGIS*-Server (localhost:5432)). Unterhalb der Serverebene befindet sich die Datenbankebene (*Databases*). In unserem Beispiel befinden sich fünf Datenbanken, wobei wir mit der Datenbank *db_hxy012* arbeiten wollen.
Unterhalb der Datenbank (hier *db_hxy012*) befinden sich drei Ebenen mit den Bezeichnungen *Catalogs*, *Schemas* und *Replication*. Hier wird nur die Ebene *Schemas* beschrieben, weil sich dort unter der Ebene *public* unsere Tabellen (*Tables*) und Sichten (*Views*) verbergen, mit denen wir arbeiten wollen.



Die Tabellen befinden sich unter dem Schema *public.*

Unter *Tables* findet man die eigenen Tabellen und zwei **Systemtabellen** von *PostGIS* mit den Namen ***geometry_columns*** und ***spatial_ref_sys***. In der Tabelle *geometry_columns* werden die Tabellen verwaltet, die mit Hilfe von *PostGIS*-Funktionen erstellt wurden. In der Tabelle *spatial_ref_sys* stehen Projektionsparameter für die Transformation der Geometrien. Bitte diese Tabellen **nicht löschen**!

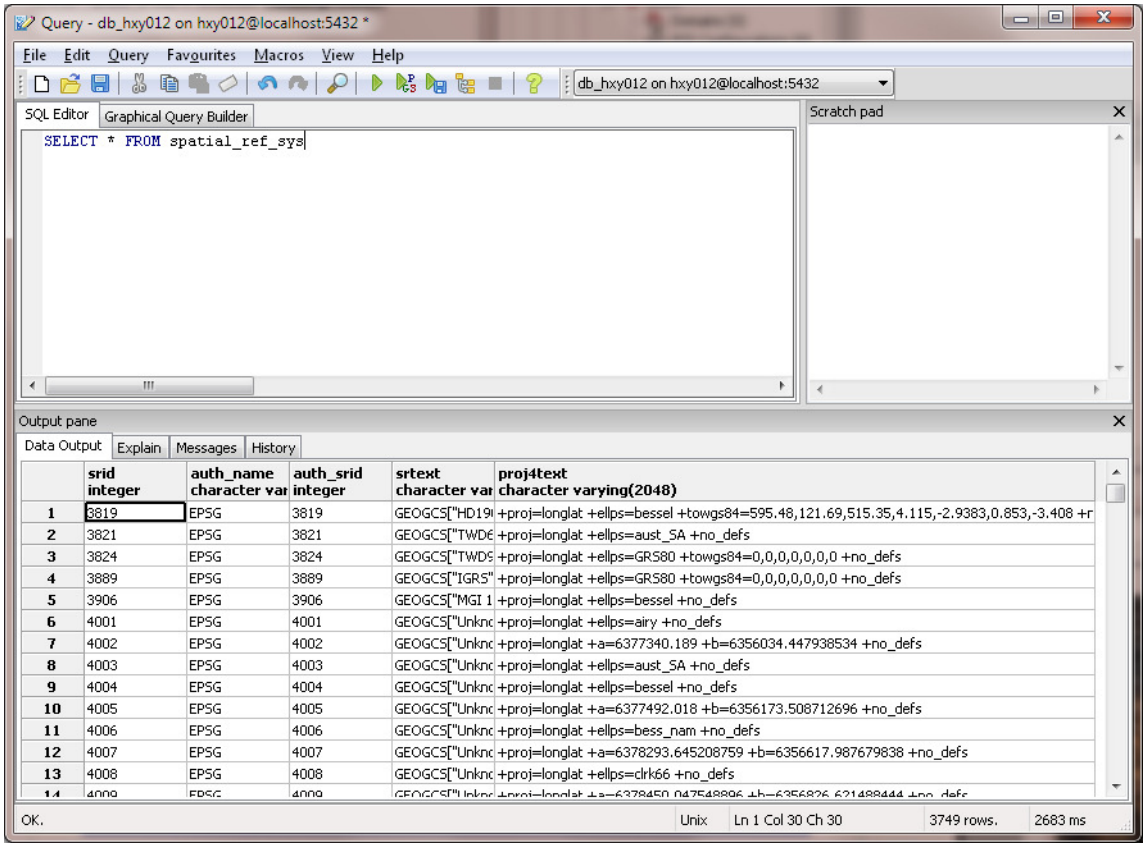
1.1.3 Das SQL-Fenster (Query Tool)

SQL-Anweisungen werden in einem separaten Fenster (*Query Tool*) eingegeben, das über ***Tools>Querytool*** oder über das Symbol  aufgerufen wird. **Bitte markieren Sie vor dem Aufruf im linken Fenster von *pgAdmin III* (*Object browser*) die Ebene *Tables*, damit Sie ein leeres SQL-Editorfenster bekommen.** Das Fenster teilt sich in ein *SQL-Editorfenster* (oberer linker Bereich), ein sogenanntes *Scratch pad* und ein Informationsfenster auf (*Output pane>Data Output*).

Im *SQL-Editorfenster* können *SQL*-Anweisungen eingegeben, korrigiert und gespeichert werden.

Im *Scratch pad* können verschiedene SQL-Anweisungen zwischengespeichert werden. Im Informationsfenster stehen die Ergebnisse der Anfrage an die Datenbank. Es können ein oder mehrere *SQL*-Anweisungen eingegeben werden, die mit ***Query>Execute*** oder über das Symbol  ausgeführt werden.

Alle Eingaben im *SQL-Editorfenster* können über ***File>Save as...*** in eine Datei gespeichert werden und mit ***File>Open*** geladen werden.



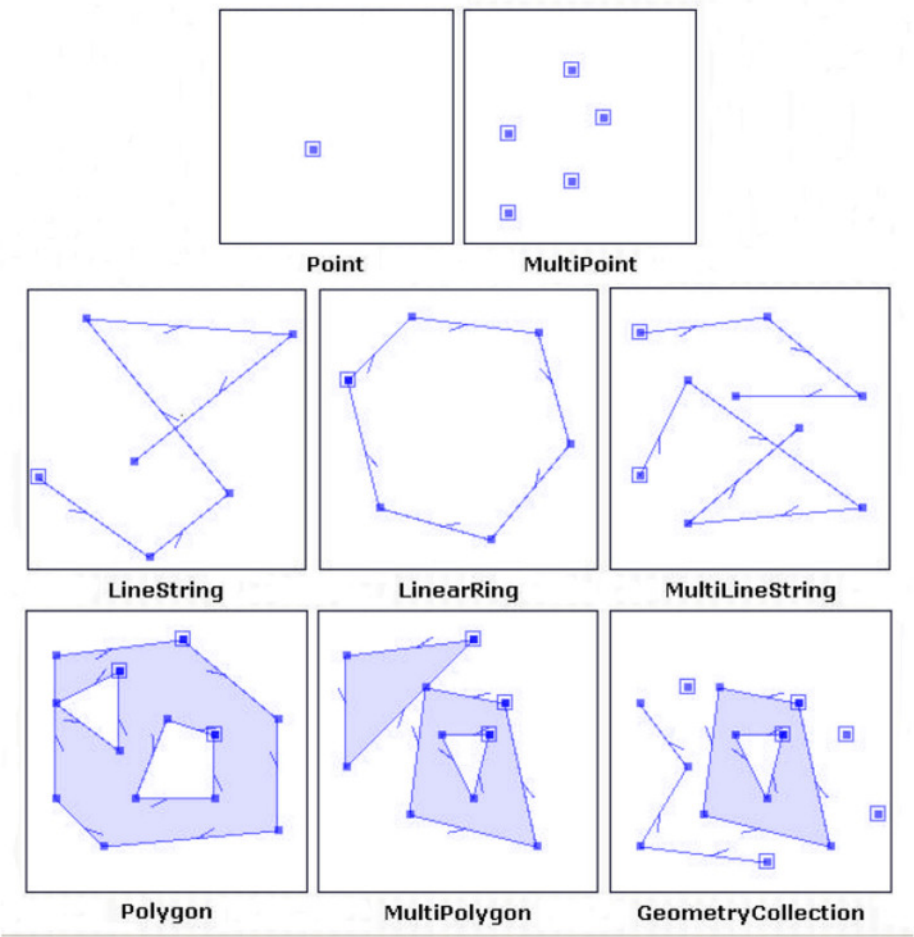
SELECT-Anfrage mit Ergebnis.

2 Funktionen zur Dateneingabe

Mit Hilfe der *PostGIS*-Funktionen können Geometrien erzeugt und auf bestehende Geometrien Analysen und Abfragen durchgeführt werden. Viele Funktionen basieren auf den *OpenGIS* Spezifikationen die in den *OpenGIS*-Dokumenten „*OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 1: Common architecture*“ und „*Part 2: SQL option*“ beschrieben sind. Anhand von Beispielen werden einige Funktionen beschrieben und angewendet. Eine vollständige Beschreibung der Funktionen befindet sich einmal im *PostGIS*-Manual und im oben erwähnten *OpenGIS*-Dokument.

2.1 Geometrien

Es sind drei verschiedene **Basis-Geometrietypen** definiert: **POINT** (Punkt), **LINESTRING** (Linienzug) und **POLYGON** (Fläche). Von diesen Basistypen sind vier weitere Geometrietypen abgeleitet: MULTIPOINT, MULTILINESTRING, MULTIPOLYGON und GEOMETRYCOLLECTION bei denen ein Objekt (Feature) aus mehreren Basis-Geometrietypen besteht.



Geometrietypen (Quelle: *JUMP*, Technical Report).

Die Geometrien können entweder im Textformat (Well-Known Text, ***WKT***) oder im binären Format (Well-Known Binary, ***WKB***) eingegeben werden. Mit einem einfachen TextEditor werden *SQL*-Statements erstellt, um Geometrien zu erzeugen. Weil das Erzeugen von Geometrien mit Hilfe des *WKB-Formats* (z.B. mit Hilfe der Funktion *ST_GeomFromWKB ( )*) nicht sehr anschaulich ist, wird hier nur auf das *WKT*-Format eingegangen (siehe auch S. 14).

Beispiel:
Ein Punkt (*Point*) wird im *WKT* Format wie folgt dargestellt: **'POINT (122.123 376.985)'**

Hinweis: Werden die Koordinaten mit Dezimalpunkt eingegeben, muss mindestens eine Nachkommastelle eingegeben werden (z.B. 10.0 und **nicht** 10.).

Eine Übersicht der Geometrietypen und die entsprechende Darstellung im *WKT*-Format gibt die folgende Tabelle:

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

Geometrietypen im *WKT*-Format (Quelle: *OpenGIS - Simple feature access – Part 1: Common architecture*).

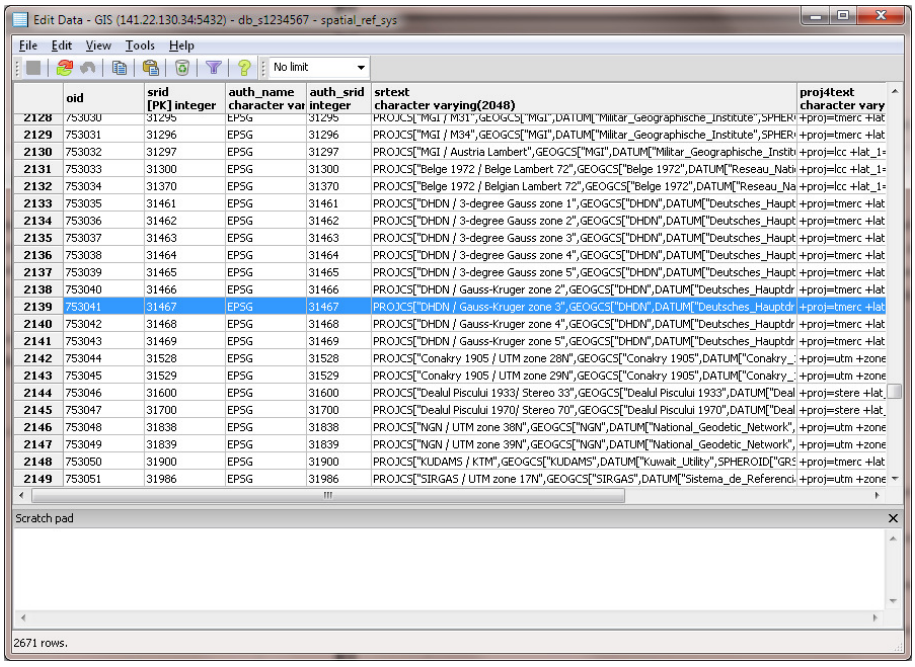
2.2 Räumliches Bezugssystem (Spatial Referencing System)

Neben den Koordinaten kann auch das räumliche Bezugssystem (Spatial Referencing System, SRS) angegeben werden. In der Tabelle *spatial_ref_sys* sind Informationen über verschiedene räumliche Bezugssysteme gespeichert, die über einen Schlüssel angesprochen werden.
Dieser Schlüssel wird **S**patial **R**eference **S**ystem **I**dentifier (**SRID**, http://en.wikipedia.org/wiki/SRID) genannt. Z.B. haben wir für eine Gauß-Krüger Projektion im 3. Streifen einen SRID von 31467. Liegt kein *SRID*-Wert vor, wird eine -1 (minus 1) gesetzt (S. 12)
Für die Schlüssel, die von der **E**uropean **P**etroleum **S**urvey **G**roup - heute **OGP** (International Association of **O**il & **G**as **P**roducers) www.epsg.org - festgelegt werden, wird die Abkürzung **EPSG** verwendet.

Beispiel:

SRID

EPSG: 4326 = Geografische Koordinaten im WGS84 Bezugssystem
EPSG: 31466 = Gauß-Krüger, 2. Streifen
EPSG: 31467 = Gauß-Krüger, 3. Streifen
EPSG: 31468 = Gauß-Krüger, 4. Streifen



	oid	srid [PK] integer	auth_name character var	auth_srid integer	srtext character varying(2048)	proj4text character vary
2128	753030	31295	EPSG	31295	PROJCS["MGI / M31",GEOGCS["MGI",DATUM["Militar_Geographische_Institute",SPHER	+proj=tmerc +lat
2129	753031	31296	EPSG	31296	PROJCS["MGI / M34",GEOGCS["MGI",DATUM["Militar_Geographische_Institute",SPHER	+proj=tmerc +lat
2130	753032	31297	EPSG	31297	PROJCS["MGI / Austria Lambert",GEOGCS["MGI",DATUM["Militar_Geographische_Instit	+proj=lcc +lat_1=
2131	753033	31300	EPSG	31300	PROJCS["Belge 1972 / Belge Lambert 72",GEOGCS["Belge 1972",DATUM["Reseau_Nati	+proj=lcc +lat_1=
2132	753034	31370	EPSG	31370	PROJCS["Belge 1972 / Belgian Lambert 72",GEOGCS["Belge 1972",DATUM["Reseau_Na	+proj=lcc +lat_1=
2133	753035	31461	EPSG	31461	PROJCS["DHDN / 3-degree Gauss zone 1",GEOGCS["DHDN",DATUM["Deutsches_Haupt	+proj=tmerc +lat
2134	753036	31462	EPSG	31462	PROJCS["DHDN / 3-degree Gauss zone 2",GEOGCS["DHDN",DATUM["Deutsches_Haupt	+proj=tmerc +lat
2135	753037	31463	EPSG	31463	PROJCS["DHDN / 3-degree Gauss zone 3",GEOGCS["DHDN",DATUM["Deutsches_Haupt	+proj=tmerc +lat
2136	753038	31464	EPSG	31464	PROJCS["DHDN / 3-degree Gauss zone 4",GEOGCS["DHDN",DATUM["Deutsches_Haupt	+proj=tmerc +lat
2137	753039	31465	EPSG	31465	PROJCS["DHDN / 3-degree Gauss zone 5",GEOGCS["DHDN",DATUM["Deutsches_Haupt	+proj=tmerc +lat
2138	753040	31466	EPSG	31466	PROJCS["DHDN / Gauss-Kruger zone 2",GEOGCS["DHDN",DATUM["Deutsches_Hauptdr	+proj=tmerc +lat
2139	753041	31467	EPSG	31467	PROJCS["DHDN / Gauss-Kruger zone 3",GEOGCS["DHDN",DATUM["Deutsches_Hauptdr	+proj=tmerc +lat
2140	753042	31468	EPSG	31468	PROJCS["DHDN / Gauss-Kruger zone 4",GEOGCS["DHDN",DATUM["Deutsches_Hauptdr	+proj=tmerc +lat
2141	753043	31469	EPSG	31469	PROJCS["DHDN / Gauss-Kruger zone 5",GEOGCS["DHDN",DATUM["Deutsches_Hauptdr	+proj=tmerc +lat
2142	753044	31528	EPSG	31528	PROJCS["Conakry 1905 / UTM zone 28N",GEOGCS["Conakry 1905",DATUM["Conakry_	+proj=utm +zone
2143	753045	31529	EPSG	31529	PROJCS["Conakry 1905 / UTM zone 29N",GEOGCS["Conakry 1905",DATUM["Conakry_	+proj=utm +zone
2144	753046	31600	EPSG	31600	PROJCS["Dealul Piscului 1933/ Stereo 33",GEOGCS["Dealul Piscului 1933",DATUM["Deal	+proj=stere +lat
2145	753047	31700	EPSG	31700	PROJCS["Dealul Piscului 1970/ Stereo 70",GEOGCS["Dealul Piscului 1970",DATUM["Deal	+proj=stere +lat
2146	753048	31838	EPSG	31838	PROJCS["NGN / UTM zone 38N",GEOGCS["NGN",DATUM["National_Geodetic_Network",	+proj=utm +zone
2147	753049	31839	EPSG	31839	PROJCS["NGN / UTM zone 39N",GEOGCS["NGN",DATUM["National_Geodetic_Network",	+proj=utm +zone
2148	753050	31900	EPSG	31900	PROJCS["KUDAMS / KTM",GEOGCS["KUDAMS",DATUM["Kuwait_Utility",SPHEROID["GRS	+proj=tmerc +lat
2149	753051	31986	EPSG	31986	PROJCS["SIRGAS / UTM zone 17N",GEOGCS["SIRGAS",DATUM["Sistema_de_Referenci	+proj=utm +zone

Die Tabelle *spatial_ref_sys*.

2.3 Tabelle mit Geometriespalte erstellen

Eine *PostGIS*-Tabelle kann Sachdaten und Geometriedaten enthalten. Die Tabellenspalten für die Sachdaten werden mit einer normalen *SQL* **CREATE**-Anweisung festgelegt. Die Spalte für die Geometriedaten muss mit der *PostGIS*-Funktion (*OpenGIS*) *AddGeometryColumn ( )* erzeugt werden, wobei es nur **eine Geometriespalte** pro Tabelle geben kann. Es sind also **zwei Schritte** notwendig, um eine Tabelle mit einer Geometriespalte zu erzeugen:

1. CREATE TABLE ...
2. SELECT *AddGeometryColumn (...)*

2.3.1 AddGeometryColumn ( )

Die Funktion *AddGeometryColumn ( )* kann mit verschiedenen Parametern aufgerufen werden:

AddGeometryColumn (Schemaname, Tabellenname, Spaltenname, SRID, Geometrietyp, Dimension)

oder wenn die Tabelle im Standardschema (public) erstellt werden soll:

AddGeometryColumn (Tabellenname, Spaltenname, SRID, Geometrietyp, Dimension)

Parameter	*Typ*	*Beschreibung*
Schemaname	VARCHAR	Das Schema, unter der die Tabelle erstellt werden soll.
Tabellenname	VARCHAR	Name der Tabelle.
Spaltenname	VARCHAR	Name der Geometriespalte.
SRID	INTEGER	Spatial Reference System Identifier, z.B. 31467 für Gauß-Krüger oder -1 wenn nicht gesetzt.
Geometrietyp	VARCHAR	z.B. 'POINT', 'LINESTRING', 'POLYGON'.
Dimension	INTEGER	Dimension der Punkte (2 oder 3).

2.3.1.1 Beispiel POINT:

Es soll eine Tabelle mit Bäumen (*baeume*) erstellt werden. Die Bäume sollen als Punkt (POINT) mit einem Primärschlüssel und dem Namen gespeichert werden. Zuerst erstellen wir eine Tabelle mit der *SQL*-Anweisung CREATE:

```
CREATE TABLE baeume ( ps INTEGER PRIMARY KEY, name VARCHAR );
```

Dann erzeugen wir die **Geometriespalte** mit der SELECT-Anweisung und der Funktion *AddGeometryColumn ( )*:

```
SELECT AddGeometryColumn ( 'baeume', 'geom', -1, 'POINT', 2 );
```

2.3.1.2 Beispiel LINESTRING:

Es soll eine Tabelle mit **Straßen** und **Straßennamen** erstellt werden. Die Straßen sollen als *LINESTRING* gespeichert werden.

```
CREATE TABLE strassen ( ps INTEGER PRIMARY KEY, name VARCHAR );
SELECT AddGeometryColumn ( 'strassen', 'geom', -1, 'LINESTRING', 2 );
```

2.3.1.3 Beispiel POLYGON:

Es soll eine Tabelle mit **Grundstücken** und **Eigentümern** erstellt werden. Die Grundstücke werden als *POLYGON* gespeichert.

```
CREATE TABLE grundstuecke ( ps INTEGER PRIMARY KEY, eigentuemer VARCHAR );
SELECT AddGeometryColumn ( 'grundstuecke', 'geom', -1, 'POLYGON', 2 );
```

2.4 Geometriedaten in Tabelle einfügen (Insert)

Nachdem die Tabellen mit

```
CREATE TABLE baeume ( ps INTEGER PRIMARY KEY, name VARCHAR );
SELECT AddGeometryColumn ( 'baeume', 'geom', -1, 'POINT', 2 );
```

oder

```
CREATE TABLE strassen ( ps INTEGER PRIMARY KEY, name VARCHAR );
SELECT AddGeometryColumn ( 'strassen', 'geom', -1, 'LINESTRING', 2 );
```

oder

```
CREATE TABLE grundstuecke ( ps INTEGER PRIMARY KEY, eigentuemer VARCHAR );
SELECT AddGeometryColumn ( 'grundstuecke', 'geom', -1, 'POLYGON', 2 );
```

erstellt wurden (S. 12), sollen Daten in die Tabelle eingefügt werden. Die Geometriedaten sollen im *WKT*-Format eingegeben werden. Dazu benötigen wir die *SQL*-Anweisung *INSERT INTO* und die *PostGIS*-Funktion **ST_GeomFromText ( )**.

2.4.1 ST_GeomFromText ( )

Die Funktion *ST_GeomFromText ( )* erzeugt ein Objekt vom Typ Geometry. Die Geometriedaten werden im *WKT*-Format übergeben.

ST_GeomFromText (text, SRID)

Parameter	*Typ*	*Beschreibung*
text	VARCHAR	Geometrie im *WKT*-Format, z.B. 'POINT (10 20)'
SRID	INTEGER	Spatial Reference System Identifier, z.B. 31467 für Gauß-Krüger oder -1 wenn nicht gesetzt.

2.4.1.1 Beispiel POINT

In die Tabelle *baeume* (siehe Kapitel 2.4) sollen der Primärschlüssel, der Baumname und die Koordinaten des Baumes P = (10, 20) eingefügt werden.

```
INSERT INTO baeume VALUES
( 1234, 'Eiche', ST_GeomFromText ( 'POINT ( 10 20 )', -1 ) );
```

Hinweis:

1. Die Koordinaten eines Punktes sind durch ein Leerzeichen getrennt!
2. Werden die Koordinaten mit Dezimalpunkt eingegeben, muss mindestens eine Nachkommastelle eingegeben werden (z.B. 10.0 und **nicht** 10.).

2.4.1.2 Beispiel LINESTRING

In die Tabelle *strassen* (siehe Kapitel 2.4) sollen der Primärschlüssel, der Straßenname und die Koordinaten der Straßenachse P1 = (30, 35); P2 = (45, 57); P3 = (60, 83) eingefügt werden.

```
INSERT INTO strassen VALUES
( 4567, 'Hofweg', ST_GeomFromText ( 'LINESTRING ( 30 35, 45 57, 60 83 )', -1 ) );
```

Hinweis:

1. Die Koordinaten eines Punktes sind durch ein Leerzeichen getrennt!
2. Die Punkte sind durch Komma getrennt.

2.4.1.3 Beispiel POLYGON

In die Tabelle *grundstuecke* (siehe Kapitel 2.4) sollen der Primärschlüssel, der Eigentümer und die Koordinaten des Grundstücks P1 = (75, 20); P2 = (80, 30); P3 = (90, 22) P4 = (85, 10); P5 = P1 = (75, 20) eingefügt werden.

```
INSERT INTO grundstuecke VALUES
( 10, 'Mayer', ST_GeomFromText ( 'POLYGON ( ( 75 20, 80 30, 90 22, 85 10, 75 20 ) )', -1 ) );
```

Hinweis:

1. Die Koordinaten eines Punktes sind durch ein Leerzeichen getrennt!
2. Die Punkte sind durch Komma getrennt.
3. Die Koordinaten des Polygons stehen in **zwei** öffnenden und **zwei** schließenden Klammern.

2.5 Tabelle mit Geometriespalte löschen

Durch die Funktion *AddGeometryColumn ( )* wird eine Datenzeile in die *PostGIS*-Systemtabelle *geometry_columns* geschrieben, die Informationen für *PostGIS* über die neu angelegte Tabelle enthält. Würde man diese neu angelegte Tabelle mit der *SQL*-Anweisung ***DROP TABLE tabellenname*** löschen, würde die Systemtabelle *geometry_columns* **nicht** aktualisiert werden. Die entsprechende Datenzeile würde nicht gelöscht werden! Eine Tabelle mit einer Geometriespalte muss daher mit der *PostGIS*-Funktion ***DropGeometryTable ( )*** gelöscht werden.

2.5.1 DropGeometryTable ( )

Die Funktion *DropGeometryTable ( )* löscht eine Tabelle mit Geometriespalte und den entsprechenden Eintrag in der Systemtabelle *geometry_columns*.

DropGeometryTable (Schemaname, Tabellenname)

oder

DropGeometryTable (Tabellenname)

Parameter	*Typ*	*Beschreibung*
Schemaname	VARCHAR	Schemaname unter der sich die Tabelle befindet
Tabellenname	VARCHAR	Zu löschende Tabelle

Beispiel:
Die Tabelle *baeume* soll gelöscht werden:

```
SELECT DropGeometryTable ( 'baeume' );
```

2.6 Tabelle im WKT-Format auflisten

Damit die Geometriespalte im *WKT*-Format dargestellt wird, muss die *PostGIS*-Funktion *ST_AsText ( )* verwendet werden. Soll der SRID-Wert mit ausgegeben werden, kann die Funktion *ST_AsEWKT ( )* verwendet werden.

Beispiel:

```
SELECT name, ST_AsText ( geom ) FROM baeume;
```

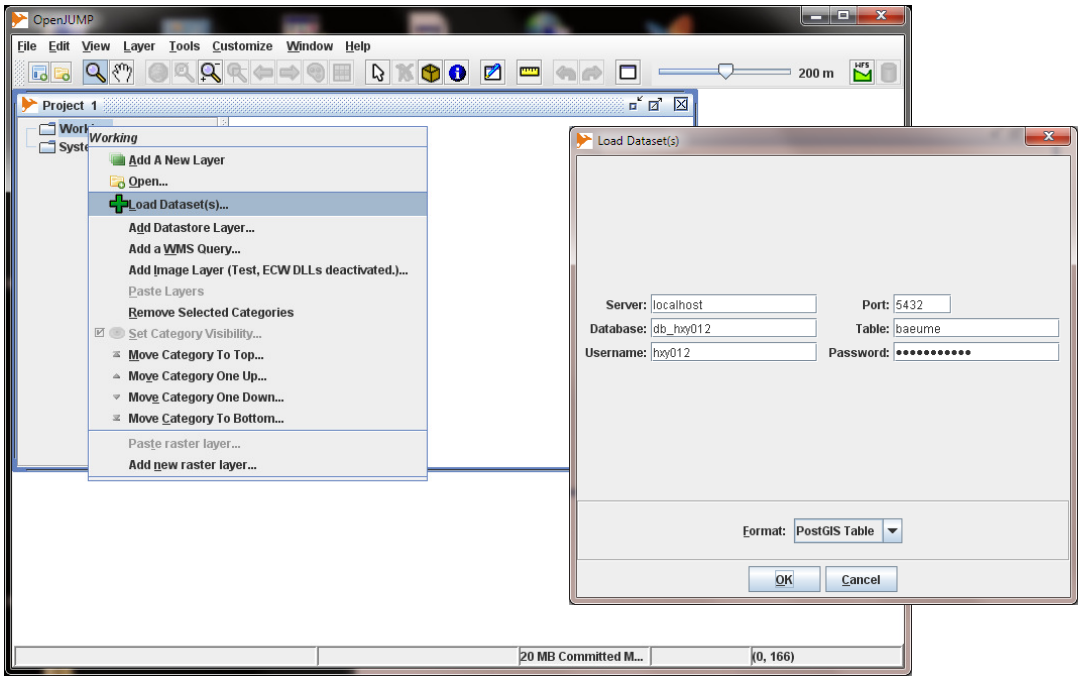
3 OpenJUMP

3.1 Lesen und darstellen von Tabellen

Nachdem die Tabelle *baeume* mit folgenden Anweisungen erstellt und mit Daten gefüllt wurde (drei Bäume),

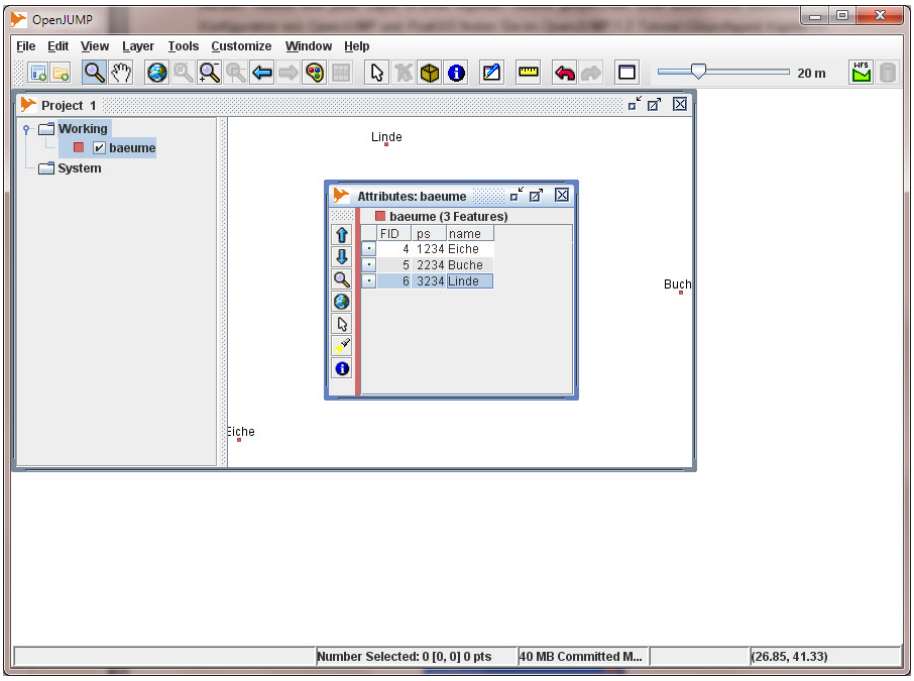
```
CREATE TABLE baeume ( ps INTEGER PRIMARY KEY, name VARCHAR );
SELECT AddGeometryColumn ( 'baeume', 'geom', -1, 'POINT', 2 );
INSERT INTO baeume VALUES ( 1234, 'Eiche', ST_GeomFromText ( 'POINT ( 10 20 )', -1 ) );
INSERT INTO baeume VALUES ( 2234, 'Buche', ST_GeomFromText ( 'POINT ( 40 30 )', -1 ) );
INSERT INTO baeume VALUES ( 3234, 'Linde', ST_GeomFromText ( 'POINT ( 20 40 )', -1 ) );
```

soll die Tabelle in *OpenJUMP* dargestellt werden. Eine ausführliche Beschreibung zur Konfiguration von *OpenJUMP* und *PostGIS* finden Sie im ***OpenJUMP 1.2 Tutorial (Grundlagen)*** Kapitel 11.
Mit RechtsKlick auf eine Karegorie und dann ***Load Dataset(s)...*** erscheint ein Fenster, in dem unter **Format: *PostGIS Table*** ausgewählt werden muss, um die Maske zur Eingabe der Datenbankserverdaten zu erhalten.



Die Tabelle *baeume* aus der Datenbank *db_hxy012* soll geladen werden.

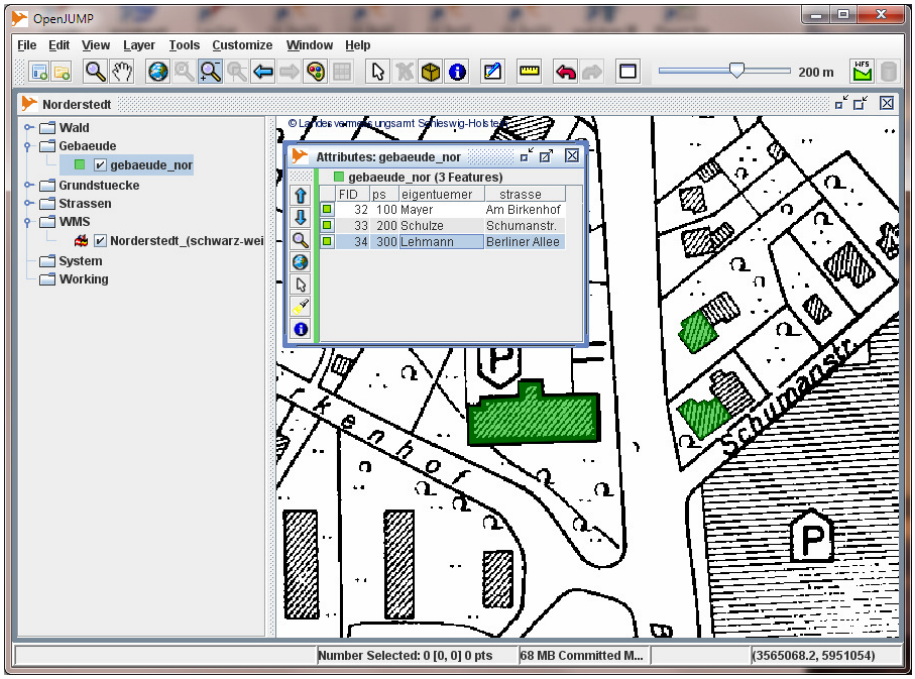
In *OpenJUMP* wird für jede Tabelle ein neuer Layer mit dem Namen der Tabelle erstellt, d.h. auf dem Layer *baeume* finden wir unsere drei Bäume wieder.
Mit RechtsKlick auf den Layernamen und ***View/Edit Attributes*** werden die Attribute der Tabelle *baeume* angezeigt.



Attribute und Geometrien der Tabelle *baeume*.

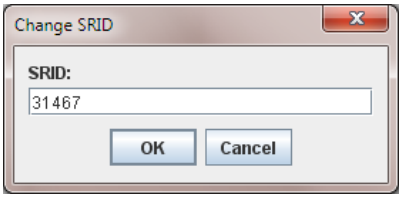
3.2 Erfassen und speichern von Daten

Geometrien und Sachdaten, die in *OpenJUMP* erfasst wurden, können in *PostGIS*-Tabellen gespeichert werden. Hierbei wird jeder *Layer* in einer eigenen Tabelle gespeichert. Eine ausführliche Beschreibung zur Konfiguration von *OpenJUMP* und *PostGIS* finden Sie im *OpenJUMP 1.2 Tutorial (Grundlagen)* Kapitel 11.
Um einen Layer in einer *PostGIS*-Tabelle speichern zu können, sollte jeder Geometrie ein eindeutiger Schlüssel, z.B. der Primärschlüssel, zugeordnet werden. Mit RechtsKlick auf den Layernamen und *View/Edit Schema* wird das Schema für die *PostGIS*-Tabelle erstellt (siehe *OpenJUMP 1.2 Tutorial*, Kapitel 6). Das Schema sollte mindestens aus einem eindeutigen Schlüssel bestehen. Mit *View/Edit Attributes* werden die entsprechenden Sachdaten erfasst (siehe *OpenJUMP 1.2 Tutorial*, Kapitel 6).



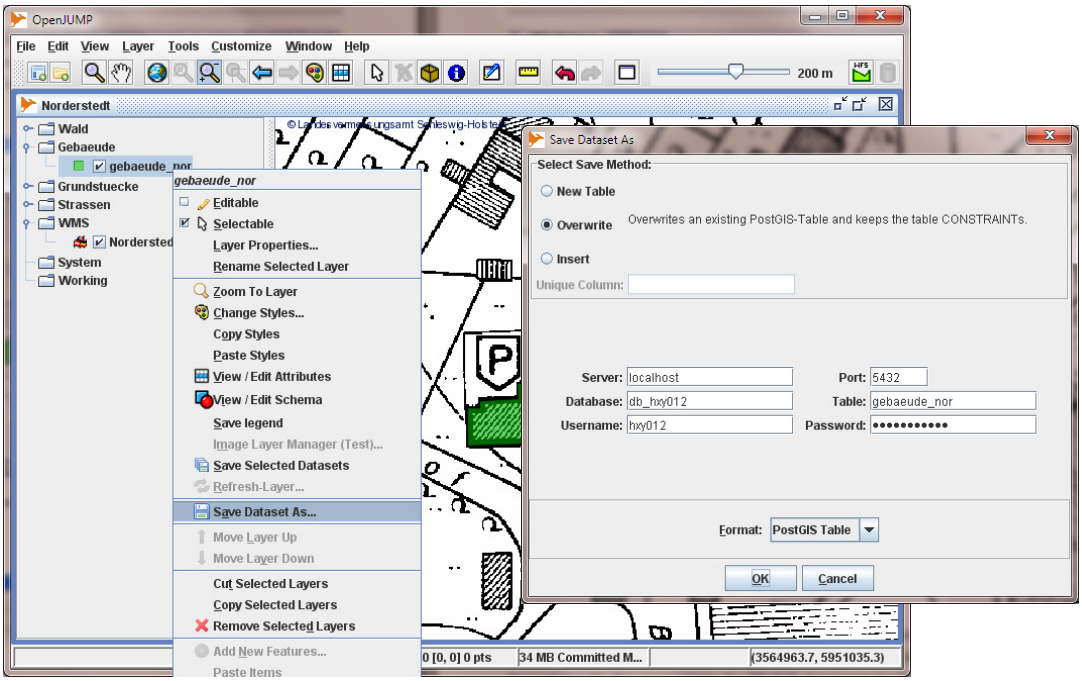
Attribute des Layers g*ebaeude_nor*.

Vor dem Sichern der Tabelle kann der Spatial Reference System Identifier (SRID) (S. 11) für den zu sichernden Layer gesetzt werden. Mit *Layer>Change SRID...* wird der Wert für den markierten Layer gesetzt.



SRID-Wert setzen.

Mit RechtsKlick auf den Layer und *Save Dataset As...* wird der **markierte** Layer in einer *PostGIS*-Tabelle gesichert. Mit der Option *Overwrite* wird ein bestehender Layer gesichert, wobei die Nebenbedingungen der Tabelle (*Constraints*) erhalten bleiben. Der zu sichernde Layername wird hinter *Table:* eingegeben (hier *gebaeude_nor*). **Achtung!** Der Layername wird **nicht** automatisch in die Maske übertragen!



Layer *gebaeude_nor* in einer *PostGIS*-Tabelle sichern.

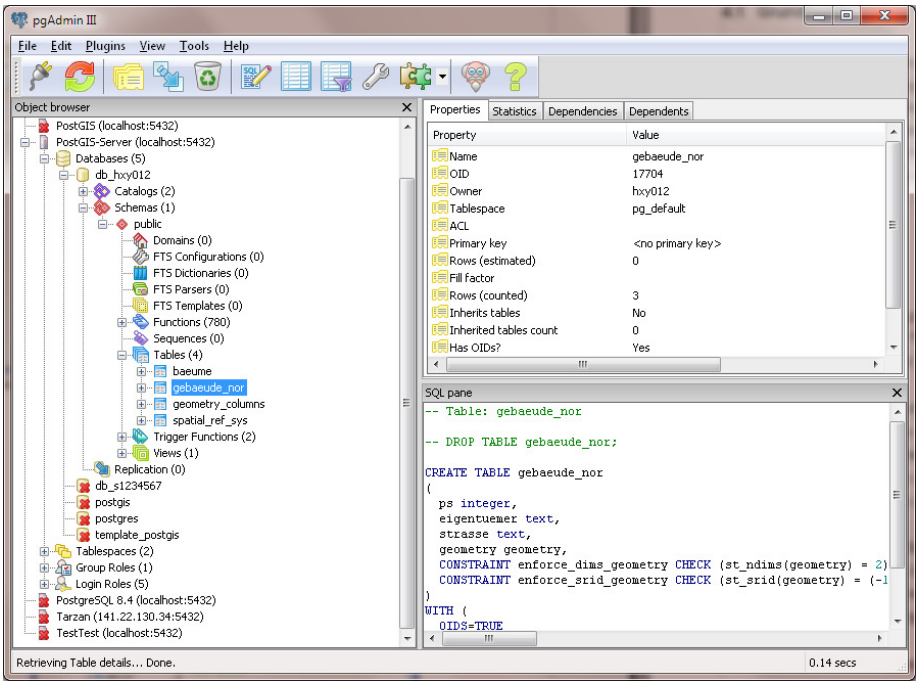


Tabelle *gebaeude_nor* der Datenbank *db_hxy012*.

4 Sichten (*Views*)

4.1 Grundlagen

In den bisherigen Beispielen wurden **alle** Datensätze einer Tabelle in *OpenJUMP* dargestellt. Es wurden z.B. **alle Bäume** der Tabelle *baeume* oder **alle Gebäude** der Tabelle *gebaeude_nor* auf jeweils einem Layer dargestellt. Für eine sinnvolle Analyse ist das nicht ausreichend! Man möchte z.B. nur die Eichen sehen oder nur die Gebäude, die Herrn oder Frau Mayer gehören. Vielleicht möchte man auch alle Eichen sehen, die auf dem Grundstück von Frau Schulze stehen. In diesem Fall benötigt man Daten aus zwei oder mehreren Tabellen. Es soll also nur eine bedingte **Sicht (*View*)** auf die Tabellen dargestellt werden! Mit der *SQL*-Anweisung CREATE VIEW ... werden solche Sichten erstellt. Das Beispiel bezieht sich auf die Tabelle *baeume* (siehe Kapitel 3.1).

Beispiel:

```
CREATE VIEW v_baeume ( Baumname ) AS
SELECT name, geom  FROM baeume
WHERE name = 'Eiche';
```

v_baeume = Name der Sicht; das Präfix ***v_*** dient nur zur besseren Unterscheidung.
(Baumname) = Neuer Spaltenname in Sicht für die erste Spalte in der Tabelle (name).
name = Spaltenname aus der Tabelle ***baeume***
geom = Spaltenname der Geometriespalte der Tabelle ***baeume***

In diesem Beispiel wird eine Sicht mit dem Namen *v_baeume* auf die Tabelle *baeume* erstellt, wo nur die Eichen von Interesse sind. Um die Tabelle in *OpenJUMP* darstellen zu können, darf natürlich die Geometriespalte (hier geom) hinter der SELECT-Anweisung der Sicht nicht fehlen. Die Sicht *v_baeume* kann man mit der SELECT-Anweisung anzeigen, wobei die Geometrie im *WKB*-Format (HEX) angezeigt wird:

```
SELECT * FROM v_baeume;
```

Data Output | Explain | Messages | History

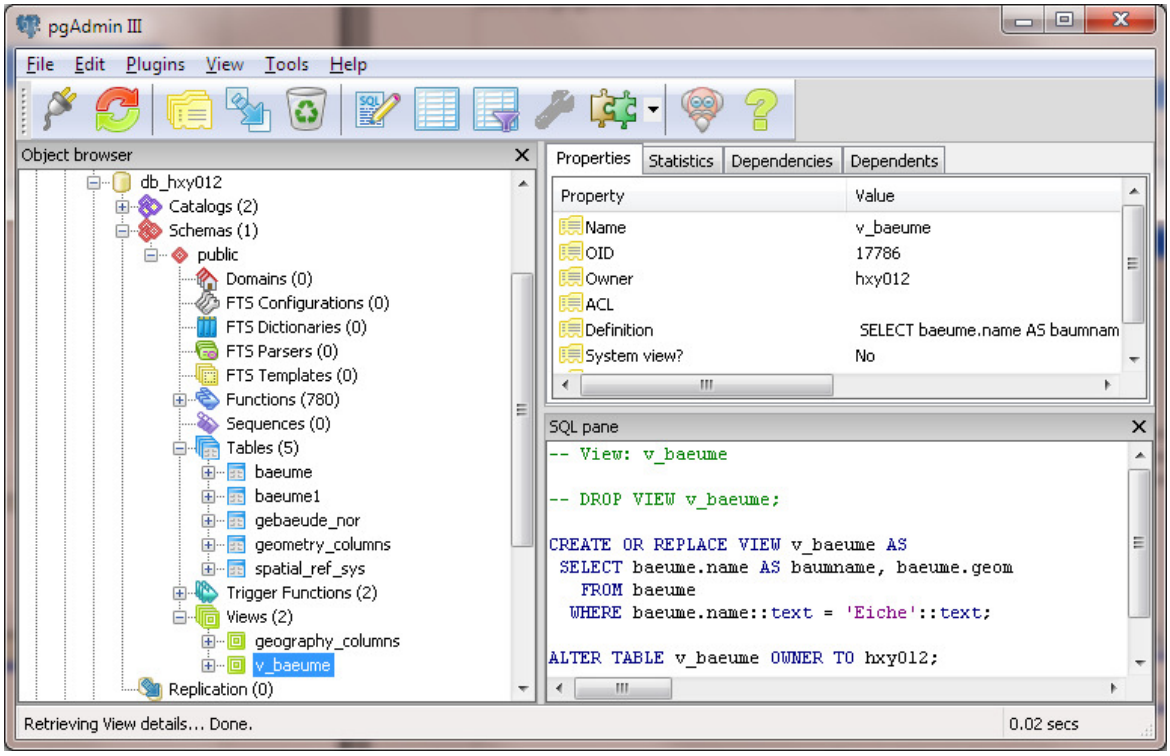
	baumname character var	geom geometry
1	Eiche	0101000000000

Ergebnis der SELECT-Anfrage auf die Sicht *v_baeume*.

Es wird nur **ein** Datensatz (Tupel) angezeigt, weil die Tabelle ***baeume*** nur **eine** Eiche enthält!

4.2 Sichten (Views) in pgAdmin III

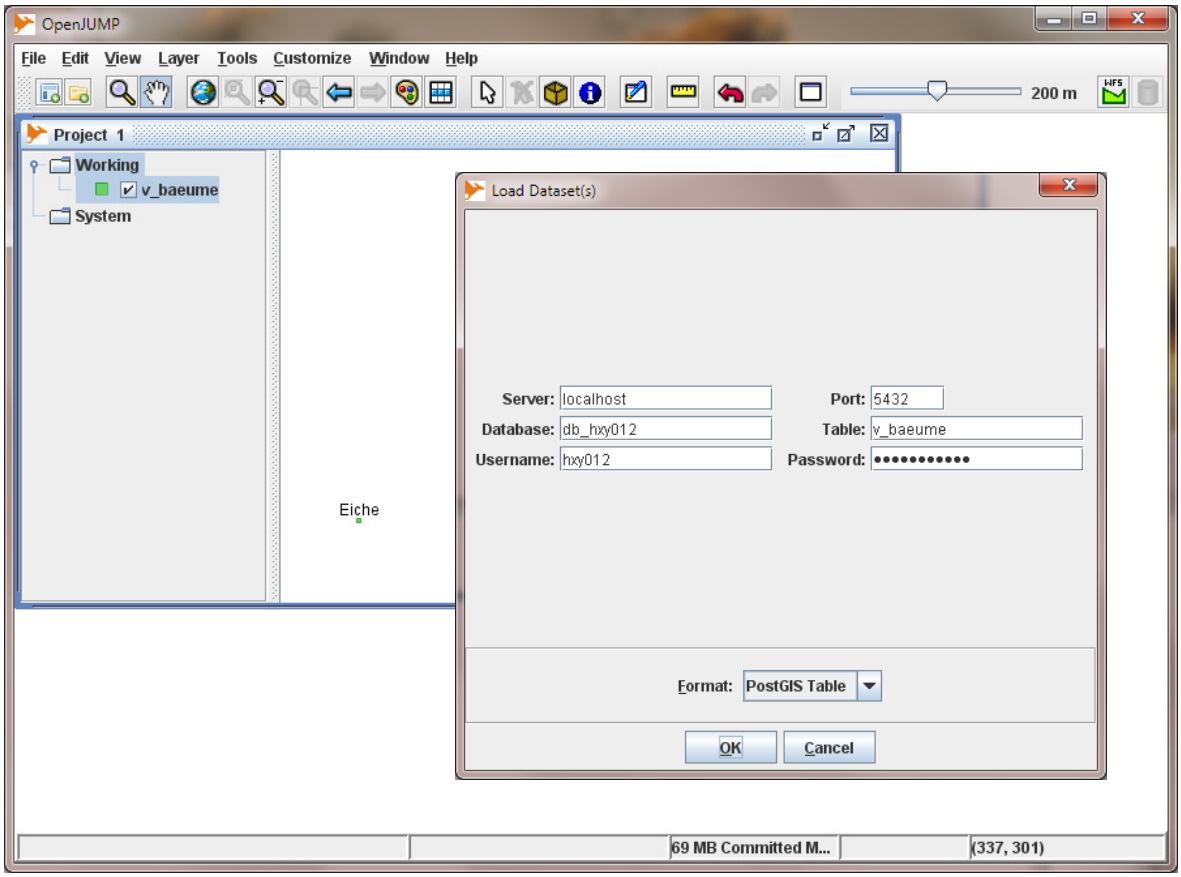
In *pgAdmin III* tauchen die Sichten (*Views*) **nicht** unter **Tables**, sondern unter **Views** auf:



Die Sicht *v_baeume* unter der Ebene *Views*.

4.3 Sichten (Views) in OpenJUMP darstellen

Sichten **mit einer Geometriespalte** können in *OpenJUMP* nur dargestellt, aber nicht erzeugt werden. Zur Darstellung muss nur in der Eingabemaske *Load Dataset(s)* anstelle des Tabellennamens der Name der Sicht eingegeben werden. In unserem Beispiel sehen wir dann nur eine einsame Eiche!



Einsame Eiche der Sicht *v_baeume*.

5 Verbund von Tabellen (*Join*)

Bisher wurden die Sachdaten und Geometrien in **einem** Datensatz (Tupel) zusammengefasst. Zu jeder Geometrie eines Gebäudes oder Grundstücks wurde z.B. der Eigentümer gespeichert. Besitzt ein Eigentümer mehrere Häuser oder Grundstücke, taucht der Name und die Adresse des Eigentümers mehrfach in unserer Tabelle auf. Die Eigentümerdaten sind redundant**!** Ändert sich z.B. die Adresse eines Eigentümers der mehrere Grundstücke besitzt, müssen alle Datensätze des Eigentümers geändert werden, was zu Fehlern führen kann. Wird ein Datensatz übersehen, ist die Tabelle inkonsistent**.**

Data Output | Explain | Messages | History

	ps integer	name character var	geom_wkt text
1	1	GlobalPlayer	SRID=31467;POLYGON((3565010.1655362
2	2	Mayer	SRID=31467;POLYGON((3565010.1655362
3	3	GlobalPlayer	SRID=31467;POLYGON((3565011.550949(
4	4	Schulze	SRID=31467;POLYGON((3564972.7593889
5	5	Lehmann	SRID=31467;POLYGON((3564972.7593889

Die Firma *GlobalPlayer* besitzt 2 Grundstücke.

Sinnvoller wäre es, die Eigentümerdaten in einer Tabelle abzulegen und die Geometrien in einer zweiten Tabelle.

Data Output | Explain | Messages | History

	ps integer	name character var	ort character var	strasse character var	telef character var
1	10	Mayer	NOR	Hauptstraße	040 1234567
2	20	Schulze	HH	Nebenstraße	040 9876655
3	30	GlobalPlayer	NOR	Schloßallee	040 5467334
4	40	Lehmann	NOR	Im Graben	040 7646383

Eigentümertabelle ohne Geometrien (*eigentuemer_nor*).

Data Output | Explain | Messages | History

	ps integer	fs integer	geom_wkt text
1	100	30	SRID=31467;POLYGON((3565010.1655362
2	200	10	SRID=31467;POLYGON((3565010.1655362
3	300	30	SRID=31467;POLYGON((3565011.550949(
4	400	20	SRID=31467;POLYGON((3564972.7593889
5	500	40	SRID=31467;POLYGON((3564972.7593889

Grundstückstabelle ohne direkte Eigentümer (*grund_fs_nor*).

Die Verknüpfung der beiden Tabellen erfolgt über Primär- (*ps*) und Fremdschlüssel (*fs*). Die Tabelle mit den Geometrien enthält als Fremdschlüssel den Primärschlüssel der Eigentümertabelle. Die Firma *GlobalPlayer* (ps = 30 der Eigentümertabelle) besitzt demnach 2 Grundstücke (ps = 100 und 300 der Grundstückstabelle)!

Sollen alle Grundstücke und die dazugehörigen Eigentümer aufgelistet werden, müssen beide Tabellen verbunden werden (*Join*):

```
SELECT e.name, e.ort, ST_AsEWKT ( g.geom ) AS geom_WKT
FROM eigentuemer_nor AS e, grund_fs_nor AS g
WHERE e.ps = g.fs;
```

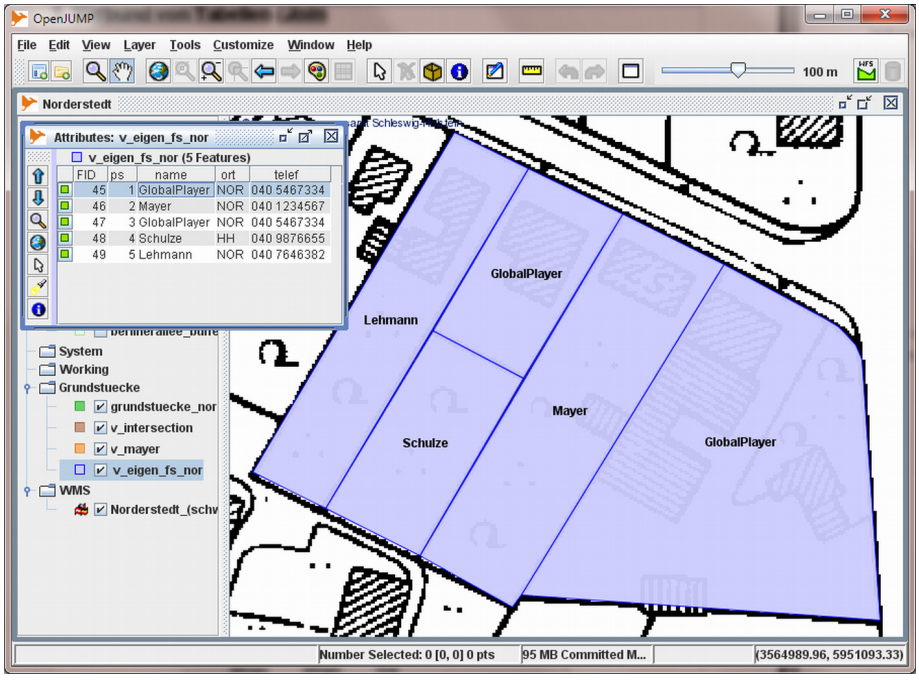
Data Output | Explain | Messages | History

	name character var	ort character var	geom_wkt text
1	GlobalPlayer	NOR	SRID=31467;POLYGON((3565010.1655362
2	Mayer	NOR	SRID=31467;POLYGON((3565010.1655362
3	GlobalPlayer	NOR	SRID=31467;POLYGON((3565011.5509490
4	Schulze	HH	SRID=31467;POLYGON((3564972.7593889
5	Lehmann	NOR	SRID=31467;POLYGON((3564972.7593889

Ergebnis der *SELECT*-Anfrage.

Sollen diese Grundstücke in *OpenJUMP* dargestellt werden, muss eine Sicht (*View*; siehe Kapitel 4) erstellt werden:

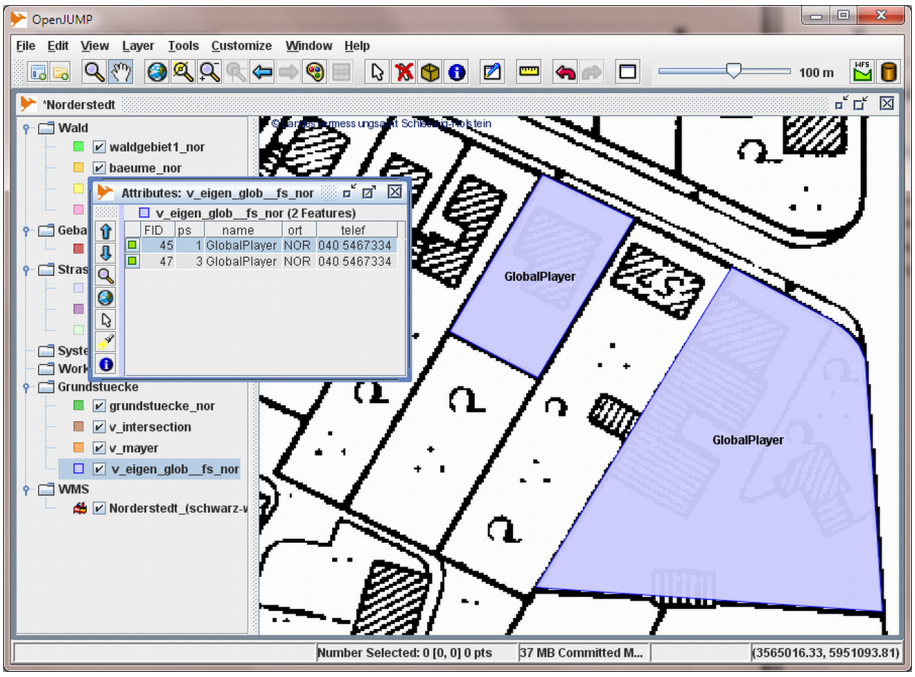
```
CREATE VIEW  v_eigen_fs_nor ( name, ort, telef, geom ) AS
SELECT e.name, e.ort, e.telef, g.geom
FROM eigentuemer_nor AS e, grund_fs_nor AS g
WHERE e.ps = g.fs;
```



Alle Grundstücke mit Eigentümer.

Sollen nur die Grundstücke der Firma *GlobalPlayer* dargestellt werden, muss noch eine Bedingung an die SELECT-Anweisung angehängt werden (**AND** e.name = 'GlobalPlayer'):

```
CREATE VIEW  v_eigen_glob__fs_nor ( name, ort, telef, geom ) AS
SELECT e.name, e.ort, e.telef, g.geom
FROM eigentuemer_nor AS e, grund_fs_nor AS g
WHERE e.ps = g.fs AND e.name = 'GlobalPlayer';
```



Die Grundstücke der Firma *GlobalPlayer*.

Hinweis: Die Funktion *ST_AsEWKT ( )* auf S. 25 dient nur zur Veranschaulichung der Geometriespalte. Für die Darstellung in *OpenJUMP* sollte die Geometriespalte direkt (ohne Funktion) in der SELECT-Anweisung stehen.

6 Berechnungsfunktionen

6.1 Längenberechnung - ST_Length ( )

Die Funktion *ST_Length ( )* berechnet die Länge eines Linienzuges (Linestring).

ST_Length (Linestring) : Double Precision

Parameter	*Typ*	*Beschreibung*
Linestring	GEOMETRY	Linienzug

Beispiel :

Gegeben: Tabelle ***strassen_nor*** mit vier Straßen:

Data Output | Explain | Messages | History

	ps integer	name character varying(255)	geom_ewkt text
1	100	Achternfelde	SRID=31467;LINESTRING(356472:
2	200	Garstedter Feldstraße	SRID=31467;LINESTRING(356468:
3	300	Kohfurth	SRID=31467;LINESTRING(356508:
4	400	Berliner Allee	SRID=31467;LINESTRING(356508:

Tabelle *strassen_nor*.

Gesucht:

1. Die **Straßenlänge** jeder Straße.
2. Die **Gesamtlänge** aller Straßen.
3. **View** mit Straßennamen, Längen und Geometrien (siehe auch Kapitel 4).

Lösung zu 1.:

```
SELECT name, ST_Length ( geom ) AS laenge FROM strassen_nor;
```

Data Output | Explain | Messages | History

	name character varying(255)	laenge double precision
1	Achternfelde	231.372389782424
2	Garstedter Feldstraße	447.888794604458
3	Kohfurth	278.927228071412
4	Berliner Allee	334.729820910642

Berechnete Straßenlängen.

Lösung zu 2.:

```
SELECT Sum ( ST_Length ( geom ) ) AS gesamtlaenge FROM strassen_nor;
```

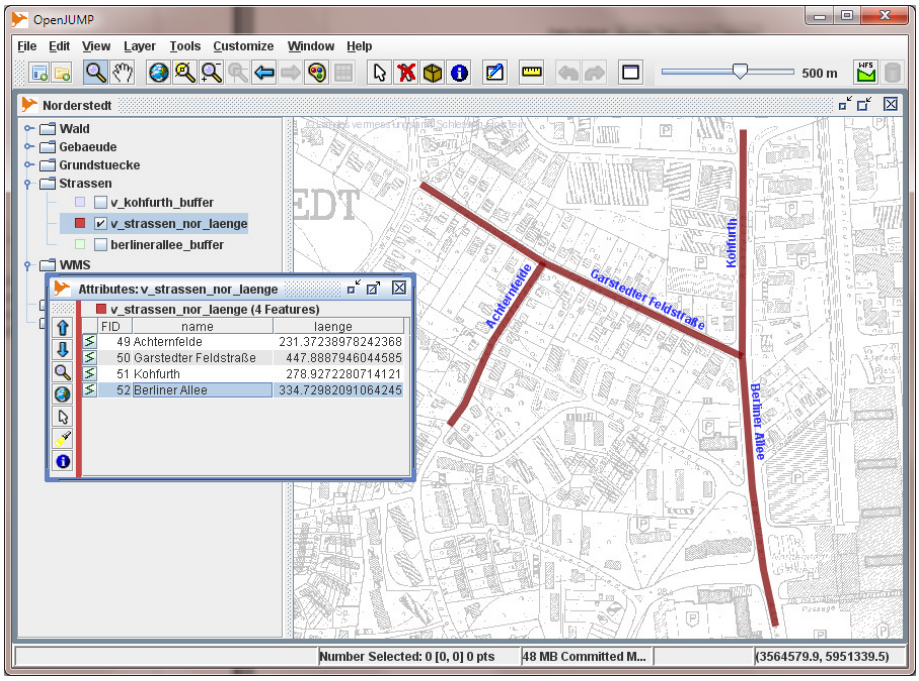
Data Output | Explain | Messages | History

	gesamtlaenge double precision
1	1292.91823336894

Gesamtlänge aller Straßen in der Tabelle.

Lösung zu 3.:

```
CREATE VIEW v_strassen_nor_laenge ( Name, Laenge)  AS
SELECT name, ST_Length ( geom ), geom FROM strassen_nor;
```



View *v_strassen_nor_laenge* in *OpenJUMP*.

6.2 Abstand - ST_Distance ( )

Die Funktion *ST_Distance ( )* berechnet den Abstand zwischen zwei Geometrien.

ST_Distance (geom1, geom2) : Double Precision

Parameter	*Typ*	*Beschreibung*
geom1	GEOMETRY	erste Geometrie
geom2	GEOMETRY	zweite Geometrie

Beispiel: Von der Schule *Lütjenmoor* sollen die Entfernungen zu den Haltestellen ermittelt werden.
Gegeben: Die Tabelle *schulen_nor* mit den Schulen und die Tabelle *haltestellen_nor* mit den Haltestellen.

Data Output | Explain | Messages | History

	ps integer	name character varying(255)	strasse character varying(255)	geom_ewkt text
1	12	Lütjenmoor	Lütjenmoor	SRID=31467;POLYGON((3565512.95585
2	14	Coppernikusgymnasium	Coppernikusstraße	SRID=31467;POLYGON((3565221.07200

Tabelle *schulen_nor*.

Data Output | Explain | Messages | History

	ps integer	haltestelle character varying(255)	linie character varying(255)	art character var	geom_ewkt text
1	1	Lütjenmoor	123	Bus	SRID=31467;POINT(3565548.766
2	2	Bf. Garstedt	123	Bus	SRID=31467;POINT(3565146.661
3	4	Ochsenzoller Straße	493	Bus	SRID=31467;POINT(3565615.677
4	5	Garstedt	U1	U-Bahn	SRID=31467;POINT(3565209.429
5	3	Bf. Garstedt	493	Bus	SRID=31467;POINT(3565150.548

Tabelle *haltestellen_nor*.

Gesucht: Die Entfernungen der Haltestellen zur Schule *Lütjenmoor*.
Lösung:

```
SELECT s.name, h.linie, ST_Distance ( s.geom, h.geom ) AS entfernung
FROM schulen_nor AS s, haltestellen_nor AS h
WHERE s.name = 'Lütjenmoor'
ORDER BY entfernung;
```

Data Output | Explain | Messages | History

	name character var	linie character var	entfernung double precision
1	Lütjenmoor	123	18.0028030938184
2	Lütjenmoor	493	158.267285017488
3	Lütjenmoor	U1	265.759108126034
4	Lütjenmoor	493	320.415452306588
5	Lütjenmoor	123	329.286722873919

Linie 123 liegt am nächsten zur Schule.

6.3 Flächenberechnung - ST_Area ( )

Die Funktion *ST_Area ( )* berechnet den Flächeninhalt eines POLYGONS.

ST_Area (Polygon) : Double Precision

Parameter	*Typ*	*Beschreibung*
Polygon	GEOMETRY	Geschlossenes Polygon

Beispiel:

Gegeben: Tabelle ***grundstuecke_nor*** mit vier Grundstücken:

Data Output | Explain | Messages | History

	ps integer	eigentuemer character var	geom_ewkt text
1	100	Mayer	SRID=31467;POLYGON((3565097.161
2	200	Schulze	SRID=31467;POLYGON((3565096.022
3	300	Lehmann	SRID=31467;POLYGON((3565148.540
4	400	Mayer	SRID=31467;POLYGON((3565153.615

Tabelle *grundstuecke_nor*.

Gesucht:

1. Die **Fläche** jedes Grundstücks.
2. Die **Gesamtfläche** aller Grundstücke der **Familie Mayer**.
3. **View** mit Eigentümer, Flächen und Geometrien der Familie Mayer.

Lösung zu 1.:

```
SELECT eigentuemer, ST_Area ( geom ) AS flaeche FROM grundstuecke_nor;
```

Data Output | Explain | Messages | History

	ps integer	eigentuemer character var	flaeche double precision
1	100	Mayer	549.646484375
2	200	Schulze	401.810546875
3	300	Lehmann	796.9921875
4	400	Mayer	586.9296875

Grundstücke mit berechneten Flächen.

Lösung zu 2.:

```
SELECT eigentuemer, Sum ( ST_Area ( geom ) ) AS gesamtflaeche
FROM grundstuecke_nor
WHERE eigentuemer = 'Mayer'
GROUP BY eigentuemer;
```

Data Output | Explain | Messages | History

	eigentuemer character var	gesamtflaeche double precision
1	Mayer	1136.576171875

Die Gesamtfläche der Grundstücke der Familie Mayer

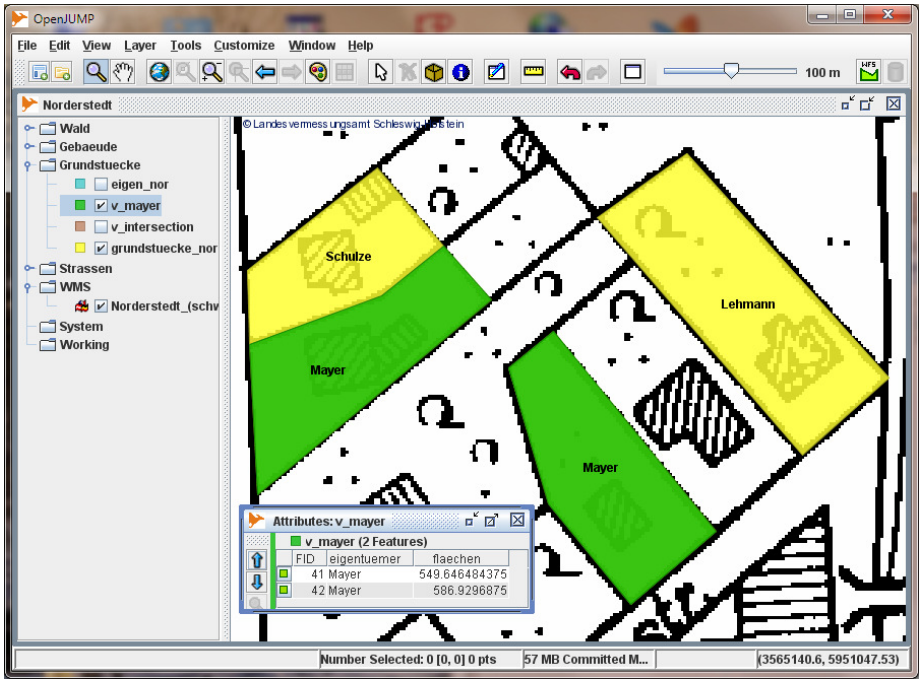
Lösung zu 3.:

```
CREATE VIEW v_mayer ( eigentuemer, flaechen, geom ) AS
SELECT eigentuemer, ST_Area ( geom ), geom
FROM grundstuecke_nor
WHERE eigentuemer = 'Mayer';
```

Data Output | Explain | Messages | History

	eigentuemer character var	flaechen double precision	geom_ewkt text
1	Mayer	549.646484375	SRID=31467;POLYGON((3565097.161'
2	Mayer	586.9296875	SRID=31467;POLYGON((3565153.615'

SELECT eigentuemer, flaechen, ST_AsEWKT (geom) AS geom_ewkt FROM v_mayer;

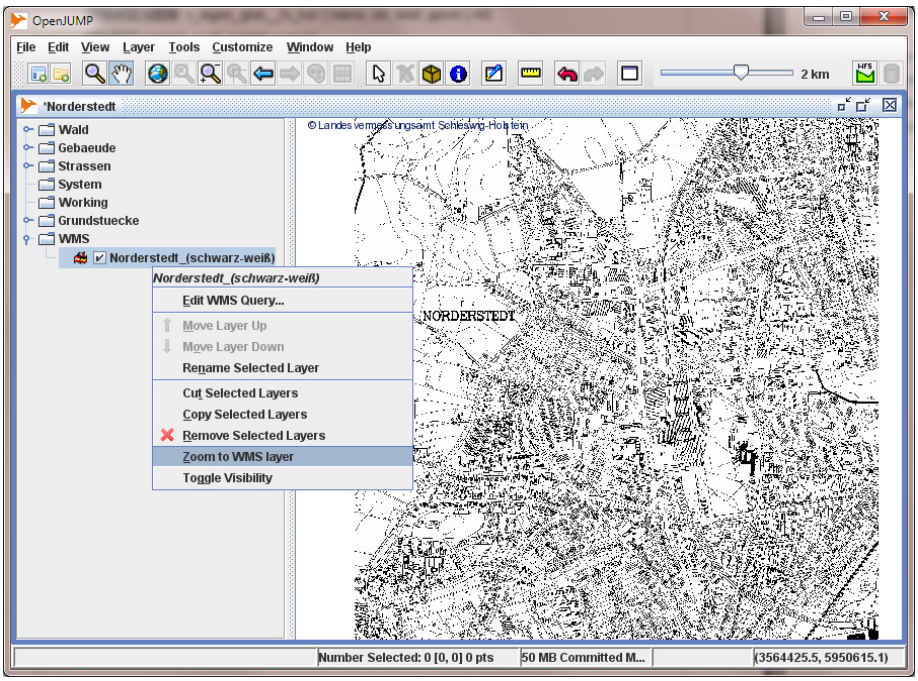


View v_mayer in *OpenJUMP*.

7 Analysefunktionen

7.1 Allgemeines

Die Geometrien für die folgenden Beispiele wurden mit Hilfe von *OpenJUMP* erstellt. Hierzu wurde eine DGK5 von *Norderstedt* als Digitalisierungsvorlage verwendet. Diese Karte wird von einem *WMS*-Server mit der *URL* *http://gis.rzcn.haw-hamburg.de/cgi-bin/mapserv.exe?map=c:/mapserver/wms/htdocs/norderstedt.map* zur Verfügung gestellt (siehe auch *OpenJUMP 1.2 Tutoria*l, Kapitel 4.5 *WMS*-Layer).



Rasterkarte als Digitalisierungsvorlage.

Hinweis: In diesem Tutorial werden nur einige wenige Analysefunktionen vorgestellt. Eine ausführliche Beschreibung aller Funktionen finden Sie im *PostGIS*-Manual: http://postgis.refractions.net/documentation/

7.2 Distanzbereich - ST_Buffer ( )

Die Funktion *ST_Buffer ( )* erzeugt ein POLYGON mit einem Abstand zu einer gegebenen Geometrie.

ST_Buffer (Geometrie, Abstand [, AnzSeg]) : POLYGON

Parameter	*Typ*	*Beschreibung*
Geometrie	GEOMETRY	POINT, LINESTRING oder POLYGON
Abstand	Double Precision	Abstand zur gegebenen Geometrie
AnzSeg	Integer	Anzahl der Segmente des berechneten Polygons

7.2.1 Beispiel POINT:

Gegeben: Tabelle *baeume_nor* mit Bäumen und Kronendurchmesser:

Data Output | Explain | Messages | History

	ps integer	name character var	krone double precis	stamm double precis	geom_ewkt text
1	100	Eiche	5.5	5.2	SRID=31467;POINT(3565117.98
2	200	Birke	2.1	0.5	SRID=31467;POINT(3565098.61
3	300	Eiche	7.8	1.2	SRID=31467;POINT(3565099.50
4	400	Linde	6.7	0.7	SRID=31467;POINT(3565124.44
5	500	Buche	5.9	1.6	SRID=31467;POINT(3565141.82
6	600	Eiche	1.7	0.4	SRID=31467;POINT(3565162.53
7	700	Birke	5.4	0.8	SRID=31467;POINT(3565159.19
8	800	Buche	10.5	2.1	SRID=31467;POINT(3565123.77
9	900	Eiche	3.4	0.5	SRID=31467;POINT(3565139.81

Tabelle *baeume_nor*.

Gesucht: View mit Geometrien der Bäume in Abhängigkeit des Kronendurchmessers.

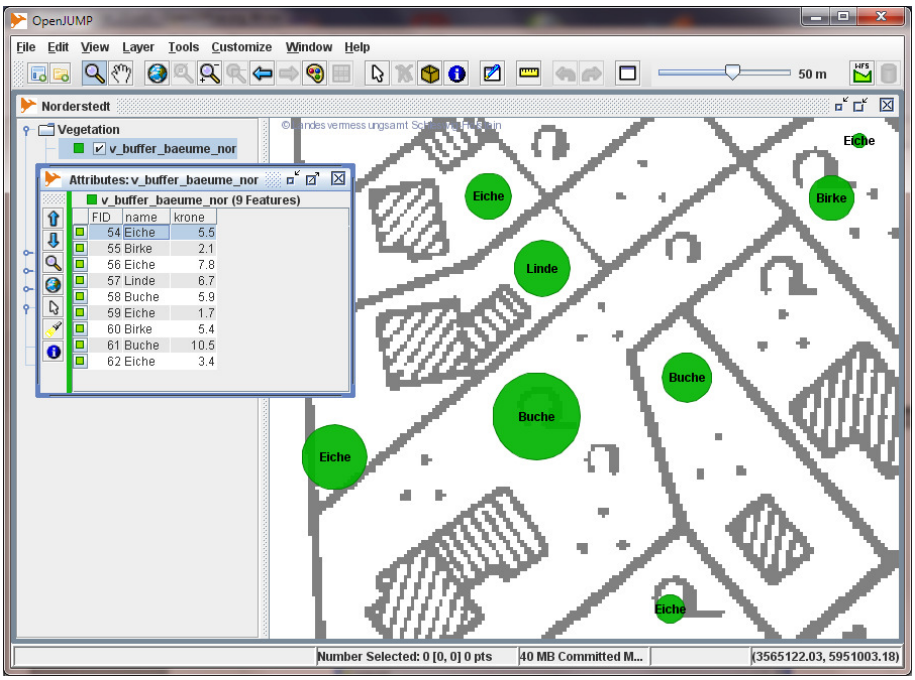
Lösung:

```
CREATE VIEW v_buffer_baeume_nor ( name, krone, geom ) AS
SELECT name, krone, ST_SetSRID ( ST_Buffer ( geom, krone / 2. ), 31467 )
FROM baeume_nor;
```

Hinweis:

1. An die Funktion *ST_Buffer ( )* wird die Geometrie der Bäume (hier *POINT*) und der Kronendurchmesser übergeben. Die von *ST_Buffer ( )* erzeugte Geometrie wird ein kreisförmiges Polygon mit dem Radius krone / 2. sein.
2. Die Funktion *ST_Buffer ( )* wird innerhalb der Funktion *ST_SetSRID ( )* aufgerufen. Die Funktion *ST_SetSRID ( )* weist der von *ST_Buffer ( )* erzeugten Geometrie den *SRID* von 31467 zu (siehe auch Kapitel 2.2),

In *OpenJUMP* können dann die Bäume in Abhängigkeit des Kronendurchmessers dargestellt werde.



Buffer um die Bäume in Abhängigkeit des Kronendurchmessers.

7.2.2 Beispiel LINESTRING:

Gegeben: Tabelle ***strassen_nor***

Data Output | Explain | Messages | History

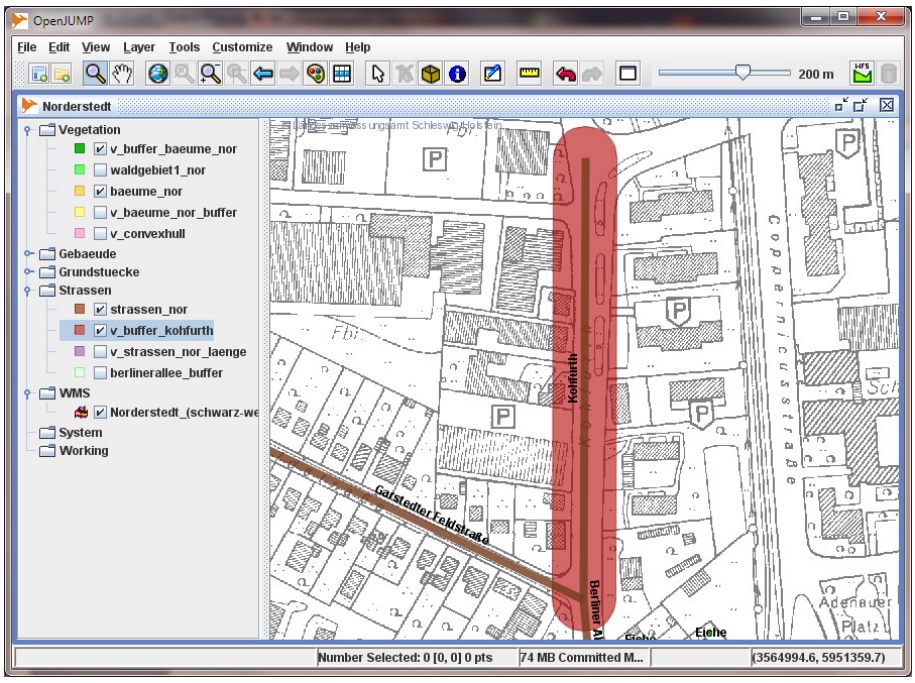
	ps integer	name character varying(2.	geom_ewkt text
1	100	Achternfelde	SRID=31467;LINESTRING(3564723.943
2	200	Garstedter Feldstraße	SRID=31467;LINESTRING(3564688.694
3	300	Kohfurth	SRID=31467;LINESTRING(3565082.563
4	400	Berliner Allee	SRID=31467;LINESTRING(3565082.563

Tabelle *strassen_nor*.

Gesucht: *View* mit Geometrie der Straße *Kohfurth*, die auf 20 m verbreitert werden soll.

Lösung:

```
CREATE VIEW v_buffer_kohfurth AS
SELECT name, ST_SetSRID ( ST_Buffer ( geom, 20. ), 31467 ) AS geom
FROM strassen_nor
WHERE name = 'Kohfurth';
```



Buffer um die Straße *Kohfurth*.

7.3 Schnittmenge - *ST_Intersection ( )*

Die Funktion *ST_Intersection ( )* berechnet die Schnittmenge zweier Geometrien. Das Ergebnis ist entweder eine neue oder eine leere (EMPTY) Geometrie. Mit der Funktion *ST_IsEmpty ( )* kann geprüft werden, ob die Geometrie leer ist.

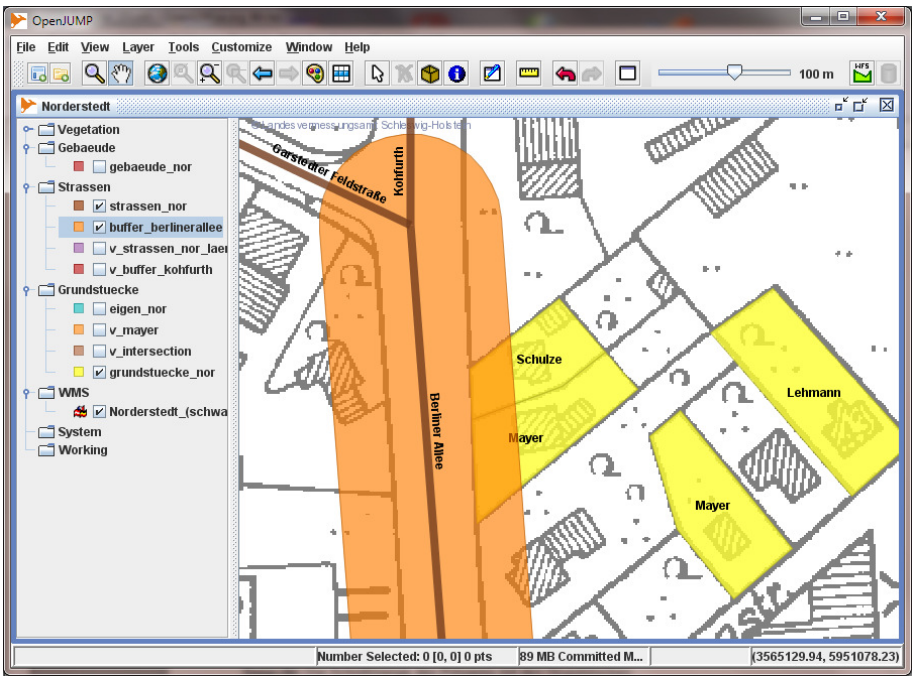
ST_Intersection (geom1, geom2) : GEOMETRY oder EMPTY

Parameter	*Typ*	*Beschreibung*
geom1	GEOMETRY	erste Geometrie
geom2	GEOMETRY	zweite Geometrie

Beispiel:
Die Straße *Berliner Allee* soll verbreitert werden (siehe Tabelle *strassen_nor* S. 35). Gesucht sind alle Grundstücke, die von der Verbreiterung betroffen sind.

Gegeben: Eine Tabelle mit dem Polygon der verbreiterten Straße (*b_BerlinerAllee*; Buffer der Straße; S. 35) und eine Tabelle mit den Geometrien der Grundstücke (*grundstuecke_nor, S. 30*).

Gesucht: Die Schnittmenge des Polygons mit den Grundstücken.



Schnittmenge zwischen verbreiterter *Berliner Allee* und betroffenen Grundstücken wird gesucht.

Lösung:

Die Tabelle *grundstuecke_nor* enthält 4 Grundstücke mit den entsprechenden Geometrien (S. 30). Die Tabelle *b_BerlinerAllee* enthält die Buffer-Geometrie. Wird eine *SELECT*-Anfrage über beide Tabellen ausgeführt (*join*), erhält man das Kreuzprodukt aus der Zeilenanzahl beider Tabellen, also 4 Ergebnisse. Zwei Geometrien müssen leer (*EMPTY*) sein, weil zwei Grundstücke nicht in der Schnittmenge liegen (S. 36)!

```
SELECT ST_Intersection ( g.geom, b.geom )
FROM grundstuecke_nor AS g, b_BerlinerAllee AS b;
```

Data Output | Explain | Messages | History

	st_asewkt text
1	SRID=31467;POLYGON((3565106.94215934 595
2	SRID=31467;POLYGON((3565096.02209994 595
3	SRID=31467;GEOMETRYCOLLECTION EMPTY
4	SRID=31467;GEOMETRYCOLLECTION EMPTY

Kreuzprodukt mit 2 leeren Geometrien.

Die leeren (EMPTY) Geometrien müssen noch durch eine *WHERE*-Klausel und der *ST_IsEmpty ( ) PostGIS*-Funktion entfernt werden.

```
SELECT ST_Intersection ( g.geom, b.geom )
FROM grundstuecke_nor AS g, b_BerlinerAllee AS b
WHERE ST_IsEmpty  ( ST_Intersection ( g.geom, b.geom )  ) = FALSE;
```

Data Output | Explain | Messages | History

	st_asewkt text
1	SRID=31467;POLYGON((3565106.94215934 595
2	SRID=31467;POLYGON((3565096.02209994 595

Die gesuchte Schnittmenge.

Hinweis: Für die tabellarische Darstellung der Geometrien im erweiterten WKT-Format wurde die Funktion *ST_AsEWKT ( )* verwendet, die in den obigen SELECT-Anweisungen fehlt!

Sollen noch die Eigentümer angezeigt und der SRID verändert werden, so sieht die *SQL*-Anweisung wie folgt aus:

```
SELECT g.eigentuemer, ST_SetSRID ( ST_Intersection ( g.geom, b.geom ), 31467 ) AS geomtext
FROM grundstuecke_nor AS g, b_BerlinerAllee AS b
WHERE ST_IsEmpty ( ST_Intersection ( g.geom, b.geom ) ) = FALSE;
```

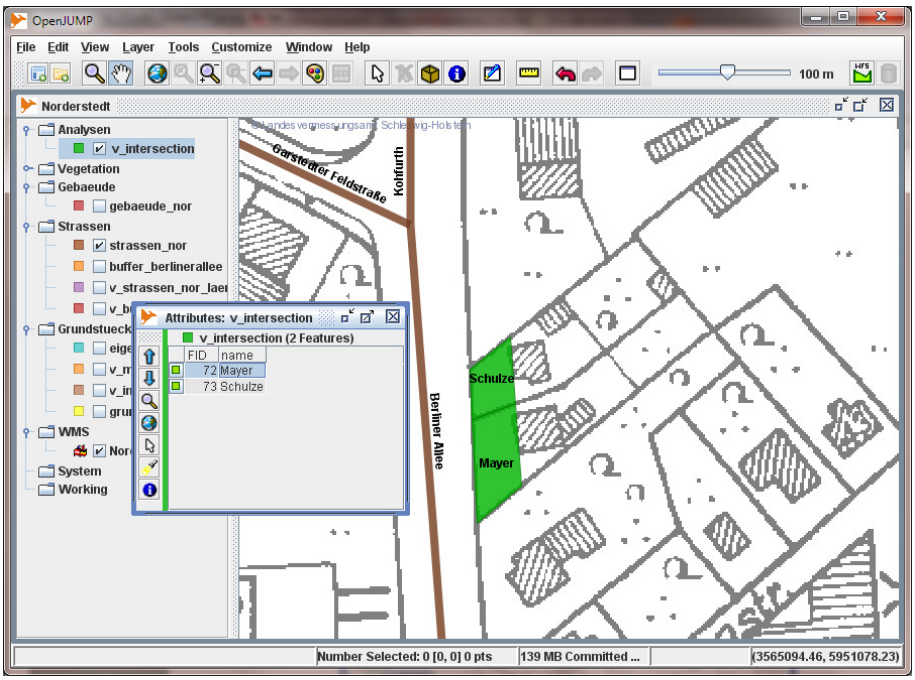
Data Output | Explain | Messages | History

	eigentuemer character var	geomtext text
1	Mayer	SRID=31467;POLYGON((3565106.94215934 595(
2	Schulze	SRID=31467;POLYGON((3565096.02209994 595:

Schnittmenge mit Eigentümer.

Zur Darstellung in *OpenJUMP* erzeugen wir eine Sicht:

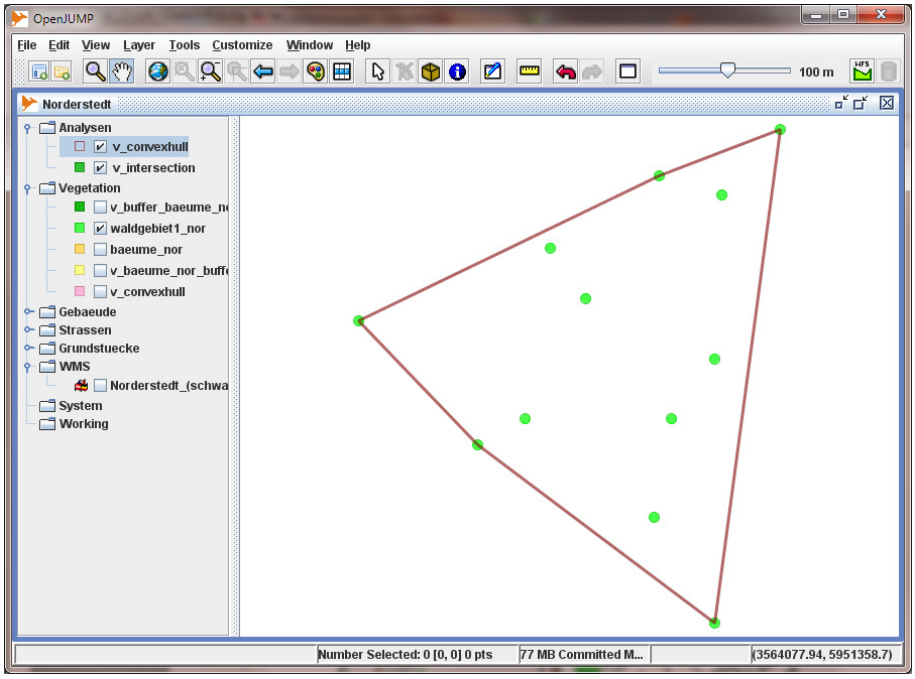
```
CREATE VIEW  v_intersection ( name, geom ) AS
SELECT g.eigentuemer, ST_SetSRID ( ST_Intersection ( g.geom, b.geom ), 31467 )
FROM grundstuecke_nor AS g, b_BerlinerAllee AS b
WHERE ST_IsEmpty ( ST_Intersection ( g.geom, b.geom ) ) = FALSE;
```



Schulze und *Mayer* sind betroffen.

7.4 Konvexe Hülle - ST_ConvexHull ( )

Die Funktion *ST_ConvexHull ( )* berechnet eine konvexe Hülle um eine oder mehrere Geometrien. Eine konvexe Hülle um eine Punktmenge ist das kürzeste Polygon, das diese Punktmenge umschließt. Spannt man z.B. ein Gummiband um die Punktmenge, so erhält man eine konvexe Hülle. Zur Veranschaulichung legen wir eine konvexe Hülle (rotes Polygon), um Geometrien vom Typ POINT:



Konvexe Hülle (rotes Polygon) um eine Punktmenge.

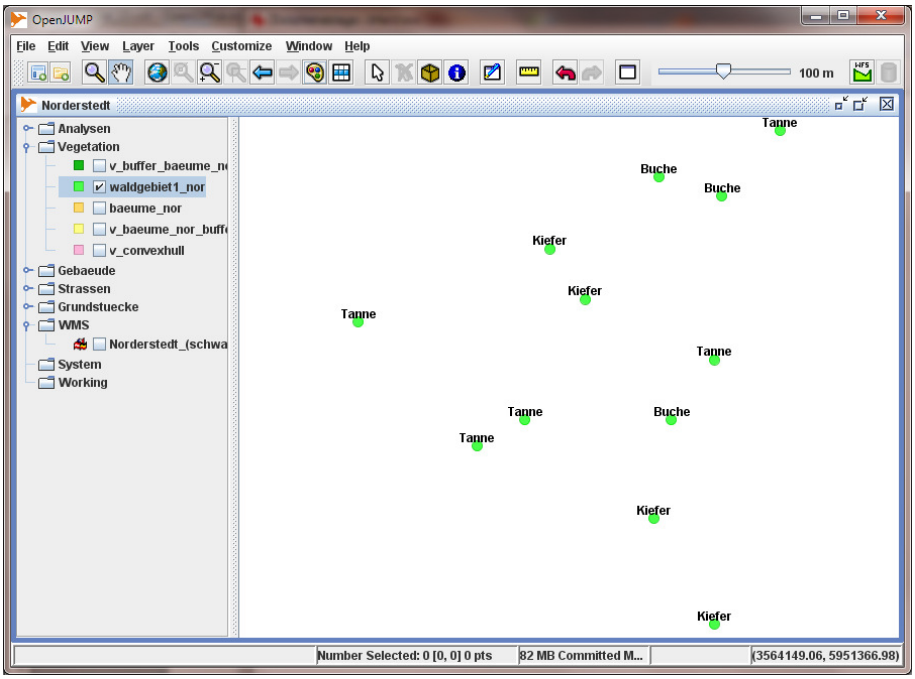
ST_ConvexHull (geom) : POLYGON

Parameter	*Typ*	*Beschreibung*
geom	GEOMETRY	Geometrie, um die eine konvexe Hülle gelegt werden soll.

Hinweis:
Soll eine konvexe Hülle um eine Menge von Punkten (POINT) gelegt werden, müssen die Punkte mit der Funktion *ST_Union ( )* zu **einer** Geometrie zusammengefasst werden.

Beispiel:

Bei einem kleinen Wäldchen in Norderstedt sind nur die Bäume in einer Tabelle erfasst. Das Wäldchen soll eingezäunt werden. Gesucht ist der kürzeste Zaun und die eingezäunte Fläche.



Kleines Wäldchen in der Tabelle *waldgebiet1_nor*.

Gegeben: Eine Tabelle *(waldgebiet1_nor*) mit den Geometrien der Bäume.

Gesucht: Konvexe Hülle um die Bäume.

Lösung:

Die Punkte der Tabelle *waldgebiet1_nor* müssen erst mit der Funktion *ST_Union (geometry set)* zu einer Geometrie zusammengefasst werden. Danach kann die konvexe Hülle berechnet werden. Dies kann alles in einer *SQL*-Anweisung geschehen.

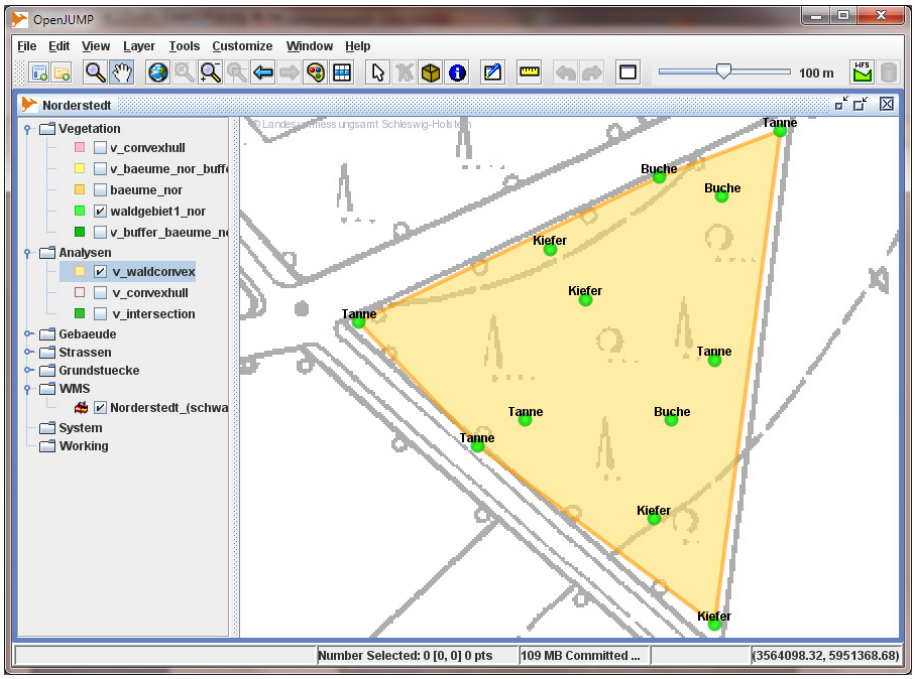
```
SELECT ST_ConvexHull  ( ST_Union ( geom )  ) FROM waldgebiet1_nor;
```

Soll die Fläche berechnet werden, muss nur die Funktion *ST_Area ( )* aufgerufen werden:

```
SELECT ST_Area ( ST_ConvexHull ( ST_Union ( geom )  )  )  AS flaeche FROM waldgebiet1_nor;
```

Zur Darstellung in *OpenJUMP* berechnet man die konvexe Hülle mit den Punkten aus der Tabelle *waldgebiet1_nor* und erzeugt daraus eine Sicht (*View*).

```
CREATE VIEW  v_waldconvex ( geom ) AS
SELECT ST_ConvexHull  ( ST_Union ( geom )  ) FROM waldgebiet1_nor;
```



Konvexe Hülle um das kleine Waldgebiet.

Leider kann man aus der konvexen Hülle (POLYGON) nicht direkt die Länge des Polygons mit der Funktion *ST_Length ( )* berechnen, sondern muss zuerst einen Linienzug (LINESTRING) aus dem POLYGON machen. Dazu dient unter anderem die Funktion *ST_Boundary ( )*. Da die Funktion *ST_Boundary ( )* ein *MULTIPOLYGON* als Argument benötigt, muss die Funktion *ST_Multi ( )* benutzt werden. Auf die Funktion *ST_Boundary ( )* und *ST_Multi ( )* wird im Moment nicht näher eingegangen.

Die Zaunlänge um das Wäldchen kann also folgendermaßen ermittelt werden:

```
SELECT ST_Length (  ST_Boundary (  ST_Multi (  geom  )  )  )  AS Zaunlaenge FROM v_waldconvex;
```

8 Abfragefunktionen

8.1 *ST_Contains ( )* und *ST_Within ( )*

- Die Funktion *ST_Contains ( )* prüft, ob **die zweite** Geometrie *geom2* in der **ersten** Geometrie *geom1* **enthalten ist**.
- Die Funktion *ST_Within ( )* prüft, ob **die erste** Geometrie *geom1* in der **zweiten** Geometrie *geom2* **enthalten ist.**

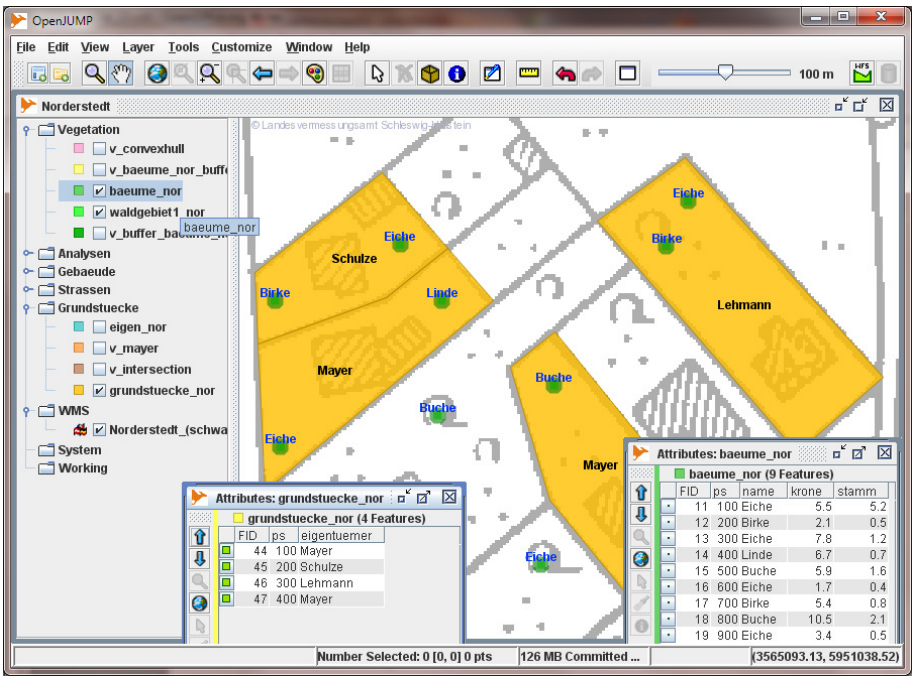
Als Ergebnis wird ein Wert vom Typ BOOLEAN zurückgegeben (TRUE, FALSE);

ST_Contains (geom1, geom2) : Boolean oder *ST_Within (geom1, geom2) : Boolean*

Parameter	*Typ*	*Beschreibung*
geom1	GEOMETRY	erste Geometrie
geom2	GEOMETRY	zweite Geometrie

Beispiel : Gesucht sind alle Bäume, die auf privaten Grundstücken stehen.

Gegeben: Die Tabelle *grundstuecke_nor* (S. 30) mit den Geometrien und Eigentümern der privaten Grundstücke und die Tabelle *baeume_nor (S. 33)* mit den Geometrien und Attributen der Bäume.



Sieben von neun Bäumen stehen auf den privaten Grundstücken.

Gesucht: Alle Bäume, die auf den Grundstücken stehen.

Lösung mit *ST_Contains ( )*:

```
SELECT gr.eigentuemer, b.name
FROM grundstuecke_nor AS gr,  baeume_nor AS b
WHERE ST_Contains ( gr.geom, b.geom ) = TRUE;
```

Hier wird geprüft, ob die Geometrien der Grundstücke (*gr.geom*) die Geometrien der Bäume (*b.geom*) enthalten.

Data Output | Explain | Messages | History

	eigentuemer character var	name character var
1	Mayer	Eiche
2	Mayer	Linde
3	Schulze	Eiche
4	Schulze	Birke
5	Lehmann	Eiche
6	Lehmann	Birke
7	Mayer	Buche

Sieben Bäume stehen auf den privaten Grundstücken.

Lösung mit *ST_Within ( )*:

```
SELECT gr.eigentuemer, b.name
FROM grundstuecke_nor AS gr, baeume_nor AS b
WHERE ST_Within ( b.geom, gr.geom ) = TRUE;
```

Hier wird geprüft, ob die Geometrien der Bäume (*b.geom*) in den Geometrien der Grundstücke (*gr.geom*) enthalten sind.

Hinweis: Es muss darauf geachtet werden, ob die Prüfung sinnvoll ist. Punktgeometrien können z.B. niemals Polygone enthalten. Die WHERE-Bedingung wäre immer FALSE!

Hinweis: In diesem Tutorial werden nur einige wenige *PostGIS*-Funktionen vorgestellt. Eine ausführliche Beschreibung aller Funktionen finden Sie im *PostGIS*-Manual: http://postgis.refractions.net/documentation/

9 Glossar

CRS: Coordiante Reference System

EPSG: European **P**etroleum **S**urvey **G**roup;
heute **OGP** (International Association of **O**il & **G**as **P**roducers) www.epsg.org
Das **O**il & **G**as **P**roducers Surveying and Positioning Committee pflegt und veröffentlicht Parameter und Beschreibungen für Koordinatenreferenzsysteme. Diese Parameter werden unter einer Kennung zusammengefasst, dem **Spatial Reference System Identifier** (**SRID**). Diese Kennungen werden z.B. in **OGC** konformen Diensten (z.B. **WMS**) und in ***PostGIS*** verwendet und ausgewertet.
(Siehe auch OGC: *„Coordinate Transformation Services")*.

Beispiel: EPSG: 4326 = Geografische Koordinaten im WGS84 Bezugssystem
EPSG: 31466 = Gauß-Krüger, 2. Streifen
EPSG: 31467 = Gauß-Krüger, 3. Streifen
EPSG: 31468 = Gauß-Krüger, 4. Streifen

Die entsprechenden Dateien mit den Datensätzen (**EPSG geodetic parameter dataset**) können von der Seite http://www.epsg.org/ geladen werden.

Feature (Objekt):

- Features sind abstrahierte Objekte der realen Welt. Zum Beispiel werden Straßen als Linienzüge, Gebäude als Flächen oder Bäume als Punkte abstrahiert und dargestellt.
  In *OpenJUMP* hat jedes Feature ein räumliches Attribut (Geometrie) und keins oder mehrere nichträumliche Attribute (non-spatial attributs, Fachdaten, Sachdaten) z.B. Straßenname, Eigentümer, Baumhöhe.

- Eine Gruppe von räumlichen Elementen, die zusammen eine Einheit der realen Welt repräsentieren. Oft synonym verwendet mit dem Ausdruck Objekt. Kann auch zu komplexen Features (Objekten), bestehend aus mehr als einer Gruppe von räumlichen Elementen, zusammengesetzt werden.
  (Lexikon der Geoinformatik, 2001)

- A geographic feature is „an abstraction of a real world phenomeon ... associated with a location relative to Earth". A feature has spatial attributes (polygons, points, etc.) and non-spatial attributes (strings, dates, numbers). (JUMP Workbench User's Guide, 2004)

GeometryCollection: Zusammenfassung von Geometrien **unterschiedlichen** Typs (S. 46).

GNU General Public License: Lizenzierung freier Software; http://www.fsf.org/licensing/licenses/gpl.html

JUMP: Unified Mapping Platform; Geografisches Informationssystem; http://www.vividsolutions.com/jump/

Mapserver: Entwicklungsumgebung für die Erstellung von Internet-Anwendungen mit dynamischen Karteninhalten; http://mapserver.gis.umn.edu/

MultiLineString: Zusammenfassung von LineString-Geometrien zu einem Objekt (S. 46).

MultiPoint: Zusammenfassung von Point-Geometrien zu einem Objekt (S. 46).

MultiPolygon: Zusammenfassung von Polygon-Geometrien zu einem Objekt (S. 46).

OGC: Open Geospatial Consortium; http://www.opengeospatial.org/
Internationales Normierungsgremium für Standards und Schnittstellen von GIS und Location Based Services (LBS) Anwendungen. Vereinigung von Firmen und Forschungseinrichtungen.

OGP: Oil & Gas Producer; http://www.ogp.org.uk/

OGP Surveying and Positioning Committee: ehemalig EPSG, http://www.epsg.org/

OpenGIS: siehe OGC; http://www.opengeospatial.org/

OpenJUMP: Geografisches Informationssystem; Erweiterung von JUMP; http://www.openjump.org/

Open Source: Quelloffenheit; http://de.wikipedia.org/wiki/Open_source

PostGIS: Erweiterung von PostgreSQL um geografische Objekte; http://postgis.refractions.net/

PostgreSQL: Objektrelationales Datenbankmanagementsystem; http://www.postgresql.org/

Refractions Research: Kanadische Firma, die JUMP mitentwickelt hat; http://www.refractions.net/

Spatial attributes: Räumliche Attribute (Punkt, Linie, Fläche).

Spatial information: Geoinformation, Rauminformation

SRID: Spatial Reference System Identifier; Kennung für Räumliches Bezugssystem

SRS: Spatial Reference System: Räumliches Bezugssystem

SVG: Scaleable Vector Graphics; vom W3C empfohlenes Grafikformat; http://www.w3.org/Graphics/SVG/
Vertex, vertices: Knoten, Eckpunkt.

Vivid Solutions: Kanadische Firma, die JUMP mitentwickelt hat; http://www.vividsolutions.com/

W3C: World Wide Web Consortium; http://www.w3.org/

Well-Known Binary (WKB): Binäre Repräsentationen für Geometrien, die in dem OpenGIS Dokument *„OpenGIS Simple Features Specification For SQL“* definiert sind.

Well-Known Text (*WKT*): Textliche Darstellung von Geometrien, die in dem OpenGIS Dokument *„OpenGIS Simple Features Specification For SQL“* definiert sind.
Ein Punkt (Point) wird z.B. als 'POINT (10 15)' dargestellt.

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

Geometrietypen im *WKT*-Format (Quelle: *OpenGIS Simple Features Specification for SQL*).

***WKB*:** siehe Well-Known Binary

***WKT*:** siehe Well-Known Text

WMS: Web Map Service; Internet-Dienst, der auf standardisierte Anfragen standardisierte Daten zur Kartenbild-Darstellung liefert. Dieser Dienst ist als OGC Standard definiert.

10 Literaturverzeichnis

Aquino, J., Davis M. (2004):
JUMP Workbench User's Guide, Vivid Solutions

Aquino, J., Kim D. (2003):
JUMP Developer's Guide, Vivid Solutions

Bill, R. (1999):
Grundlagen der Geo-Informationssysteme, Band 1, Wichmann Verlag

Bill, R. (1999):
Grundlagen der Geo-Informationssysteme, Band 2, Wichmann Verlag

Bill R., Zehner M. L. (2001):
Lexikon der Geoinformatik, Wichmann Verlag

Eisentraut, P. (2003):
PostgreSQL Das Offizielle Handbuch, mitp-Verlag Bonn

Gemeinschaftsprojekt von CCGIS und terrestris:
Praxishandbuch WebGIS mit Freier Software
http://www.terrestris.de/hp/shared/downloads/Praxishandbuch_WebGIS_Freie_Software.pdf

Lake, R., Burggraf D. S.,Trninic M., Rae L. (2004):
Geography Mark-Up Language (GML), John Wiley & Sons, Ltd

Lange, N. (2002):
Geoinformatik in Theorie und Praxis, Springer-Verlag Berlin Heidelberg New York

OGC (2003):
OpenGIS Geography Markup Language (GML) Implementation Specification, Open GIS Consortium

OGC (2001):
OpenGIS Implementation Specification: Coordinate Transformation Services, Open GIS Consortium

OGC (2004):
Web Map Service (WMS), Version: 1.3, Open GIS Consortium

OGC (2006)

OpenGIS Implementation Specification for Geographic information, Open GIS Consortium

- Simple feature access – Part 1: Common architecture
- Simple feature access – Part 2: SQL option

Refractions Research (2005):

PostGIS Manual

RRZN (2004):

SQL Grundlagen und Datenbankdesign, Regionales Rechenzentrum / Universität Hannover

The PostgreSQL Global Development Group (2005):

PostgreSQL 8.1.0 Documentation

11 Linksammlung

JUMP	http://www.vividsolutions.com/jump/
MapServer	http://www.umn-mapserver.de/
OGP Surveying & Positioning Committee	http://www.epsg.org/
Open Geospatial Consortium	http://www.opengeospatial.org/
OpenJUMP	http://www.openjump.org/
PIROL, Fachhochschule Osnabrück	http://www.al.fh-osnabrueck.de/jump-download.html
PostGIS	http://postgis.refractions.net/
PostgreSQL	http://www.postgresql.org/

Stichwortverzeichnis

Das ist das Letzte!