Hochschule Neubrandenburg
Studiengang Geoinformatik

Erstellung eines Tutorials über PostGIS in der aktuellen Entwicklungsversion 2.0

Bachelorarbeit
Zum Erlangen des akademischen Grades
Bachelor of Engineering (B.Eng.)

Erstellt von: Daniel Vogel
URN:nbn:de:gbv:519-thesis 2011-0531-3

Erstprüfer: Prof. Dr.-Ing. Ernst Heil
Zweitprüfer: Dipl.-Inform. J. Schäfer

Kurzfassung:

PostGIS erweitert das objekt-relationale Datenbanksystem PostgreSQL um die Möglichkeit der Speicherung und Verarbeitung von Geodaten. Beide Projekte stellen derzeit eines der fortschrittlichsten und am weitesten verbreiteten Geodatenbanksysteme im Bereich freier Software dar.

Im Rahmen dieser Arbeit wurde ein Tutorial erstellt, das die Installation, den Aufbau sowie Nutzungsmöglichkeiten des PostGIS Systems darstellen soll.

Das Tutorial basiert auf der derzeit noch in der Entwicklung befindlichen Version 2.0, da diese wichtige neue Funktionen insbesondere die Unterstützung für Rasterdaten beinhaltet.

Abstract:

PostGIS extends the object-relational database PostgreSQL to include the ability to store and perform processing on geographic data. Both projects form one of the most advanced geodatabase systems in the area of free software.

The Aim of this thesis is to create a tutorial covering the installation, the structure as well as several use cases of the PostGIS system.

The tutorial is based on the current development version of PostGIS-2.0 since it introduces several important new features like support for raster data.

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Neubrandenburg, den

Daniel Vogel

Inhaltsverzeichnis

1 Einstieg

1.1 Entstehung

PostGIS ist eine Erweiterung für das Datenbank Management System PostgreSQL, die die Speicherung, Abfrage, Analyse und Manipulation von Geographischen Daten ermöglicht.

Die Entwicklung des Systems begann als Forschungsprojekt von Refractions Research Inc, einem Beratungsunternehmen mit Spezialisierung auf Datenbanken aus Victoria, British Columbia im Jahr 2001. Inzwischen ist PostGIS ein Open Source Projekt, das neben Refractions Research Inc von einer Community sowie von verschiedenen Unternehmen genutzt und weiterentwickelt wird.

Die PostgreSQL Datenbank, auf der PostGIS aufbaut, ist ein umfangreiches objekt-relationales Datenbank Management System. Das Projekt ging vor 15 Jahren aus einem Forschungsprojekt an der Berkeley Universität hervor und bezeichnet sich selbst als die "fortschrittlichste Open Source Datenbank der Welt" („The world's most advanced open source database“). Als wichtigste Eigenschaften von PostgreSQL, die es für die Entwicklung von PostGIS besonders geeignet machten, werden die Unterstützung von selbst definierten Datentypen, als auch die gute Dokumentation inklusive Programmcode Beispielen angegeben. [vgl. 02 und 03]

Zukünftige Entwicklungsziele sind die Unterstützung der OpenGIS Standards, der weitere Ausbau der Topologieunterstützung für komplexe Topologische Konstrukte und die Entwicklung von webbasierten und Desktopoberflächen als Schnittstelle zu PostGIS.

1.2 Organisation

PostGIS ist ein Open Source Projekt zu dem jeder, Unternehmen und Privatpersonen, Beiträge leisten kann.

Das wichtigste Organisationsorgan ist das sogenannte Project Steering Committee (PSC). Dieses besteht aus einer Reihe von Personen, die unter anderem die generelle Entwicklungsrichtung, den Veröffentlichungszyklus neuer Versionen, die Dokumentation und Abstimmungen koordinieren. Das PSC leistet außerdem allgemeinen Nutzersupport und prüft und akzeptiert Programmänderungen und Erweiterungen aus der Entwicklergemeinschaft.

1.3 Datenmodell

PostGIS ermöglicht die Speicherung und Verarbeitung von Geometrie-, Raster- und Topologiedaten.

Seit der Version 0.9 wird der OGC Standard "Simple Features for SQL" (SFS) vollständig unterstützt. SFS definiert GIS Objekttypen, Funktionen, um diese zu bearbeiten, und eine Reihe von Tabellen für Metadaten. Die beiden Metadatentabellen sind die „GEOMETRY_COLUMNS“ Tabelle und die Tabelle SPATIAL_REF_SYS, die räumliche Bezugssysteme anhand sogenannten SRIDs und dazugehörigen textuellen Beschreibungen verwaltet.

Als Austauschformat für "Simple Features for SQL" konforme Daten werden die OpenGIS Standards WKT, „well known text“, und WKB, „well known binary“ eingesetzt. OGC SFS konforme Daten enthalten nur 2d Geometrien.

PostGIS bietet zusätzlich eigene erweiterte Geoobjekte mit sogenannten 3dm, 3dz und 4D Koordinaten und der Möglichkeit, den SRID in dem Austauschformat zu integrieren. Diese können in den Formaten EWKT und EWKB, wobei das E für „extended“ bzw. erweitert steht, ausgegeben und eingelesen werden.

Aktuell ist es das Ziel der PostGIS Entwicklung, ebenfalls den Standard „SQL-MM Part 3“ umzusetzen. Dieser Standard legt Geoobjekte und Funktionen für Geometrie, Raster und Topologie Daten fest. Die Umsetzung von SQL-MM ist in der vorliegenden Version von PostGIS noch nicht abgeschlossen.

1.4 Konventionen und Struktur dieses Tutorials

Konventionen

Im Tutorial wird zwischen normalem Text und Konsolenanweisungen bzw. SQL Befehlen unterschieden. Die Darstellungen verhalten sich dabei wie folgt:

- „Dies ist die Darstellungsweise für normalen Text. Er wird für alle Erläuterungen und für Hinweise zu den Code-Beispielen verwendet. “

- `„Dies ist die Darstellungsweise für Shell- und SQL-Befehle.“`

Die SQL-Beispiele sollten über jeden SQL-Client, der die hier verwendeten Versionen von PostgreSQL unterstützt, durchführbar sein.

Für die Erstellung wurde der in zur PostgreSQL gehörende Client psql und in beträchtlichem Maße pgAdmin III verwendet. Die Ausgaben der SQL-Befehle wurden mithilfe von pgAdmin III in das *.csv Format konvertiert und zur besseren Darstellung in einem Tabellenkalkulationsprogramm optisch aufbereitet.

Wenn SQL Anweisungen im normalen Text erwähnt werden, werden diese ganz in Großbuchstaben geschrieben.

Beispiel:

“Die SELECT Anweisung gibt Datensätze zurück, die in der FROM Anweisung eine virtuelle Tabelle darstellen.“

Das primäre System für die Einrichtung und Erstellung des Tutorials ist ein Debian Linux. Auf diesem System wurde ein unprivilegierter Nutzer namens **„nutzer“** eingerichtet. Daneben werden ebenfalls Nutzer der PostgreSQL Datenbank namens **„postgres“** und **„pgnutzer“** sowie der Systemadministator **„root“** verwendet.

Diese Rollen werden im normalen Text wie im obigen Absatz fett und in Anführungszeichen dargestellt.

Sämtliche dieser Nutzer verwenden das Passwort: „PostGISVM“.

Quellenangaben werden generell standardmäßig für einzelne Abschnitte angebracht. Im Fall der offiziellen PostGIS-2.0 Dokumentation wird davon allerdings abgewichen, da der überwiegende Teil der Ausführungen auf den Informationen dieser Quelle basiert und eine explizite Auszeichnung die Aussagekraft der Quellenangaben verschlechtern würde. Ähnlich verhält es sich mit der offiziellen PostgreSQL Dokumentation, die als Nachschlagewerk für die Erstellung der SQL Abfragen verwendet wurde und nur in spezifischen Fällen extra erwähnt wird.

Struktur

Das Tutorial besteht größtenteils aus den beiden Hauptbereichen der Vektor- und Rasterverarbeitung mit PostGIS.

Jeder dieser Bereiche besteht aus vier Teilbereichen:

1. Erläuterungen über die grundlegenden Datenstrukturen und Funktionsweisen des Systems

2. Erläuterungen bezüglich der verschiedenen Möglichkeiten Daten des entsprechenden Bereiches in der Datenbank zu speichern und wieder zu laden
3. Eine Auflistung und Gruppierung der von PostGIS zur Verfügung gestellten Funktionen
4. Anwendungsbeispiele mit Erläuterungen

1.5 Visualisierung mittels gvSIG

PostGIS unterstützt die Speicherung und Manipulation von Geodaten, ist als Datenbankerweiterung aber nicht dafür ausgestattet, Daten zu visualisieren. Aufgrund der freien Verfügbarkeit und großen Verbreitung des Systems stehen derzeit Schnittstellen zu PostGIS in verschiedenen Desktop-GIS Programmen zur Verfügung.

Im Rahmen dieses Tutorials wurde für die Visualisierung bestimmter Ergebnisse und das Digitalisieren von Daten direkt nach PostGIS das Open Source GIS gvSIG in der Version 1.11 verwendet. Andere freie Desktop GIS Anwendungen, die Schnittstellen zu PostGIS besitzen, sind zum Beispiel QuantumGIS, Udig oder OpenJUMP.

2 Installation und Konfiguration

2.1 Installationshinweise

Die Tutorial-Datenbank wurde auf einer minimalen Debian Installation aufgesetzt. Die dargestellten Installationsschritte können sich daher auf anderen Distributionen unterscheiden, sind aber vom Grundprinzip darauf übertragbar.

Die Schritte in diesem Abschnitt erfordern generell Administrationsrechte und sind darauf ausgerichtet, in der Rolle **"root"** ausgeführt zu werden. Wenn ein anderer Nutzer verwendet werden soll, wird gesondert darauf hingewiesen.

Grundsätzlich wurde versucht, für die Installation der einzelnen Abhängigkeiten wenn möglich auf die offiziellen Paketquellen mittels apt-get zurückzugreifen.

PostGIS 2.0 liegt zum Zeitpunkt der Erstellung nur als Entwicklungsversion vor und baut auf aktuellen Versionen der GEOS und GDAL Bibliotheken auf. Diese Programmpakete müssen deshalb als Quelltext heruntergeladen und auf dem Tutorialsystem übersetzt werden.

Schrittfolge für die Installation

Installation bzw. Aktualisierung der nötigen Programmpakete zum Übersetzen

```
apt-get install gcc
apt-get install g++
apt-get install gnulib
apt-get install make
```

Installation bzw. Aktualisierung der notwendigen Abhängigkeiten

PostgreSQL:

```
apt-get install postgresql postgresql-client postgresql-contrib \ postgresql-doc
postgresql-plpython-8.4 postgresql-server-dev-8.4
```

proj4:

```
apt-get install libproj-dev libproj0 proj-bin
```

libxml2:

```
apt-get install libxml2 libxml2-dev libxml2-utils
```

GDAL:

Download der Quellcodedatei unter : http://download.osgeo.org/gdal/gdal-1.8.1.tar.gz

Alternativ ist die hier verwendete GDAL Version im Anhang des Tutorials zu finden.

Entpacken an gewünschten Ort

```
cd /Pfad_zum_Quellcode_Ordner
./configure
make
make install
```

gdal Python bindings:

```
apt-get install python-gdal
```

GEOS:

Download der Quellcodedatei unter : http://download.osgeo.org/geos/geos-3.3.0.tar.bz2

Alternativ ist die hier verwendete GEOS Version im Anhang des Tutorials zu finden.

Entpacken an gewünschten Ort

```
cd /Pfad_zum_Quellcode_Ordner
```

```
./configure
make
make install
```

Installation bzw. Aktualisierung der optionalen Abhängigkeiten:

Die optionalen Abhängigkeiten werden für die korrekte Funktionsweise von PostGIS und somit für die Durchführung des Tutorials nicht benötigt.

Cunit:

Ermöglicht nach der Übersetzung die Ausführung der im PostGIS Verzeichnis enthaltenen Unit-Tests.

```
apt-get install libcunit1 libcunit1-dev
```

ant:

Wird für die Erstellung der PostGIS Java Treiber benötigt

```
apt-get install ant1.7
```

docbook:

Wird für die Erstellung der Dokumentation aus den Quellen benötigt.

```
apt-get install xsltproc
```

imagemagick:

Wird für die Erstellung der Bilder der Dokumentation benötigt.

```
apt-get install imagemagick
```

Übersetzen und Installieren von PostGIS 2.0SVN:

- Download der Quellcodedatei unter :http://postgis.refractions.net/download/postgis-2.0.0SVN.tar.gz

Alternativ ist die hier verwendete PostGIS-2.0 Revision 7774 im Anhang des Tutorials zu finden.

- Entpacken an gewünschten Ort

```
cd /Pfad_zum_Quellcode_Ordner
./configure --with-gui --with-raster --with-topology
```

Ergebnis der Konfiguration:

```
-------------- Compiler Info -------------
  C compiler:           gcc -g -O2
  C++ compiler:         g++ -g -O2

 -------------- Dependencies --------------
  GEOS config:          /usr/local/bin/geos-config
  GEOS version:         3.3.0
  GDAL config:          /usr/local/bin/gdal-config
  GDAL version:         1.8.0
  PostgreSQL config:    /usr/bin/pg_config
  PostgreSQL version:   PostgreSQL 8.4.8
  PROJ4 version:        47
  Libxml2 config:       /usr/bin/xml2-config
  Libxml2 version:      2.7.8
  PostGIS debug level:  0
 -------------- Extensions --------------
  PostGIS Raster:       0.1.6d
  PostGIS Topology:     0.1.1
```

```
-------- Documentation Generation --------
 xsltproc:             /usr/bin/xsltproc
 xsl style sheets:
 dblatex:
 convert:              /usr/bin/convert
```

Übersetzung und Installation:

```
make
make comments
make install
make comments-install
```

Installieren anderer Anwendungen:

Pgadmin 3:

Pgadmin3 ist ein graphisches Frontend für die PostgreSQL Datenbank und kann anstelle des Kommandozeilenprogramms psql verwendet werden.

```
apt-get install pgadmin3 pgadmin3-data
```

gvSIG:

- Download des binären gvSIG Installers, "gvSIG-1_11-1305-final-lin-i586-withjre-j1_5.bin" von der gvSIG Website unter:

  http://www.gvsig.org/web/projects/gvsig-desktop/official/gvsig-1.11/downloads

- Shellanweisungen – auszuführen mit der Identität von "nutzer":

- ```
  cd /Pfad zur Datei/
  chmod 700 gvSIG-1_11-1305-final-lin-i586-withjre-j1_5.bin
  ./gvSIG-1_11-1305-final-lin-i586-withjre-j1_5.bin
  ```

Hinweis: Die Zeichen "./" legen ausdrücklich fest, dass die ausführbare Datei im aktuellen Verzeichnis gestartet wird. Dies ist notwendig, da die Ausführung ansonsten verweigert wird. Grund sind Sicherheitsmechanismen des Betriebssystems.

- Abarbeiten der geführten grafischen Installation.

Im Tutorialsystem wurde die Java Runtime (JRE) des gvSIG Installationsprogramms genutzt.

Hinweise:

In der Testinstallation war das Starten von gvSIG über die Desktop Verknüpfung nicht möglich, sodass der Start über folgenden Befehl aus einem Terminal Fenster heraus durchgeführt werden muss.

```
Sh /Installationspfad von gvSIG/gvSIG.sh
```

Diese Installationsmethode wurde gewählt, da zum Erstellungszeitpunkt keine aktuelle Version von gvSIG in den offiziellen Paketquellen zur Verfügung stand. Im Allgemeinen ist die Installation von Programmen aus den Paketquellen vorzuziehen.
```

2.2 Einrichten der Tutorial-Datenbank

Festlegen eines Passworts für den Benutzer postgres in der Datenbank:

```
sudo –u postgres psql –c "ALTER ROLE postgres WITH PASSWORD 'PostGISVM';"
```

Dieser Schritt ist auf dem verwendeten Debian System nötig, damit sich Drittanwendungen wie pgadmin oder gvSIG an der Datenbank anmelden können. Die Vergabe eines Passworts über die psql Nutzererstellung funktioniert hier ausdrücklich nicht. Dies gilt auch für andere neu angelegte Benutzer. Diese Einschränkung bezieht sich auf das verwendete Testsystem und kann auf anderen Linux Distributionen oder neueren Debian Versionen behoben sein.

Erstellen einer Datenbank und laden der PostGIS Skripte:

```
sudo –u postgres createdb postgis_tutorial_template

sudo –u postgres createlang plpgsql postgis_tutorial_template

sudo –u postgres psql –d postgis_tutorial_template –f
/usr/share/postgresql/8.4/contrib/postgis–2.0/postgis.sql
sudo –u postgres psql –d postgis_tutorial_template –f
/usr/share/postgresql/8.4/contrib/postgis–2.0/spatial_ref_sys.sql
sudo –u postgres psql –d postgis_tutorial_template –f
/usr/share/postgresql/8.4/contrib/postgis–2.0/postgis_comments.sql
sudo –u postgres psql –d postgis_tutorial_template –f
/usr/share/postgresql/8.4/contrib/rtpostgis–2.0/rtpostgis.sql
sudo –u postgres psql –d postgis_tutorial_template –f
/usr/share/postgresql/8.4/contrib/postgis–2.0/raster_comments.sql
sudo –u postgres psql –d postgis_tutorial_template –f
/usr/share/postgresql/8.4/contrib/postgis–2.0/topology.sql
```

Laden der aus Postgis 2.0 entfernten veralteten Funktionen:

Dieser Schritt ist nur nötig, wenn Anwendungen wie die im Tutorial verwendete Version von gvSIG, die noch als „deprecated“ markierte Funktionen verwenden, auf die Datenbank zugreifen sollen.

```
sudo –u postgres psql –d postgis_tutorial_template –f
/usr/share/postgresql/8.4/contrib/postgis–2.0/legacy.sql
```

Festlegen der Datenbank als Template für weitere PostGIS Datenbanken:

```
sudo –u postgres psql –c "update pg_database SET datistemplate='true' WHERE
datname='postgis_tutorial_template';"
```

PostgreSQL Datenbanken können als template definiert werden, sodass die Tabelle mit all ihren Objekten vom hier verwendeten Programm "createdb" kopiert werden können. Dies kann dafür genutzt werden, um für spezielle Anforderungen angepasste Datenbankmuster zu erstellen, beispielsweise wie hier für PostGIS.

Anlegen eines neuen Postgres Nutzers

```
sudo –u postgres createuser pgnutzer –l –W
sudo –u postgres psql –c "ALTER ROLE pgnutzer WITH PASSWORD  'PostGISVM';"
```

Anlegen einer neuen Datenbank aus dem Template für den neuen Nutzer und Vergeben der nötigen Berechtigungen

```
sudo –u postgres createdb postgis_tutorial –O pgnutzer –T
postgis_tutorial_template

sudo –u postgres psql postgis_tutorial –c " GRANT all ON geometry_columns,
geography_columns,spatial_ref_sys,raster_columns,raster_overviews TO pgnutzer";
```

Einrichten der Rechteverwaltung

Damit die Beispiele im Tutorial so ausgeführt werden können, müssen drei Dateien im Zusammenhang mit der PostgreSQL Rechteverwaltung editiert werden.

.pgpass

Im Verzeichnis des Betriebssystemnutzers, hier „/home/nutzer“ befindet sich nach der Installation von PostgreSQL die Datei .pgpass. Diese sollte zunächst leer sein und muss für das Tutorialsystem um den folgenden Eintrag ergänzt werden, wofür ein einfacher Texteditor verwendet werden kann.

```
localhost:5432:*:pgnutzer:PostGISVM
#Eintragungen erfolgen nach dem Muster:
#Hostname:Port:Datenbankname:Nutzername:Passwort
```

pg_ident.conf und pg_hba.conf

Im Verzeichnis der PostgreSQL Konfigurationsdateien, hier „/etc/postgresql/8.4/main“ befinden sich die Dateien pg_ident.conf und pg_hba.conf.

Die Datei wird um folgenden Eintrag erweitert, der aufgrund besserer Übersichtlichkeit unter der ebenfalls dargestellten Kommentarzeile, die in der Datei bereits vorhanden sein sollte, eingefügt werden sollte.

```
# MAPNAME     SYSTEM-USERNAME    PG-USERNAME
tutorialnutzer nutzer pgnutzer
```

Ähnlich dazu muss die pg_hba.conf. wie folgt editiert werden:

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
# "local" is for Unix domain socket connections only
local   all         all                               ident map=tutorialnutzer
```

2.3 Vorstellung der verwendeten Demodatensätze

Die im Tutorial verwendeten Demodaten bestehen aus einer Reihe von ESRI Shapefiles, einem Mosaik aus RGB Orthophotos und einer Fernerkundungsaufnahme mit 6 Kanälen. Die Daten beziehen sich auf das Gebiet von und um Neubrandenburg und sind georeferenziert im ETRS89 System in der Zone 33 der UTM Abbildung, was dem EPSG 25833 entspricht. Ausnahme ist ein Datensatz im Bezugssystem EPSG 28403, der als Grundlage für eine Transformation in ein anderes Bezugssystem durch PostGIS genutzt wird.

2.3.1 ESRI Shapefiles

Die verwendeten Shapefiles besitzen größtenteils ein Textattribut mit der Bezeichnung „LABEL“ und ein numerisches Attribut „CAT“. Beide dienen in der Regel der Bezeichnung derselben Eigenschaft. Einige

Dateien besitzen andere oder zusätzliche Attribute, diese werden aber im Verlauf des Tutorials nicht verwendet.

(jeweils als Satz aus den entsprechenden *.shp, *.dbf, *.shx und *.prj Dateien):

Flächendaten

- flaechennutzung – stellt die verschiedenen amtlichen Flächennutzungen innerhalb des Neubrandenburger Stadtgebietes dar
- gebaeude – stellt einen Teil der Gebäude im Neubrandenburger Stadtgebiet dar und enthält zusätzliche Datenfelder
- gemeinden. - stellt die Gemeinden um Neubrandenburg dar
- gewaesser – stellt die stehenden Gewässer im Stadtgebiet von Neubrandenburg dar
- siedlungsformen – stellt die Siedlungsformen innerhalb der bebauten Gebiete Neubrandenburgs dar
- stadtgebiete – stellt die Stadtgebiete Neubrandenburgs dar
- topographie – stellt die Topographie in einem großen Gebiet um Neubrandenburg dar

Liniendaten

- B_Str – stellt die Bundesstraßen im Gebiet um Neubrandenburg dar und enthält zusätzliche Datenfelder
- L_Str - stellt die Landesstraßen im Gebiet um Neubrandenburg dar und enthält zusätzliche Datenfelder
- bahn - stellt die Bahnschienen im Gebiet um Neubrandenburg dar
- fluesse - stellt die Flüsse im Gebiet um Neubrandenburg dar
- ortsumgehung – stellt die geplante Ortsumgehungsstraße für Neubrandenburg dar
- radwanderwege – stellt die Radwanderwege im Gebiet um Neubrandenburg dar
- stadtgrenze – stellt die stadtgrenze von Neubrandenburg dar
- verkehr – stellt Verkehrswege innerhalb Neubrandenburgs dar, die Daten hier überschneiden sich mit den Land- und Bundesstraßen in B_Str und L_Str

Daten im EPSG 28403

- B_str_krassowski – ist inhaltlich gleich zu B_Str

Punktdaten

- hotels – stellt die Standorte von Hotels in Neubrandenburg dar
- sparkasse – stellt die Standorte von Sparkassen Filialen in Neubrandenburg dar
- touri – stellt die Standorte von touristischen Einrichtungen in Neubrandenburg dar

2.3.2 Rasterdaten

- 333825934.jpg; 333825936.jpg; 333845934.jpg; 333845936.jpg; 333865934.jpg; 333865936.jpg - Eine Sammlung von Orthophotos, die ein Gebiet im Zentrum Neubrandenburgs abbilden

- neubrandenburg.tif - eine LANDSAT Aufnahme des Gebietes von Neubrandenburg und Umgebung mit 6 Frequenz Kanälen

3 Vektorgeometrien

Vektorgeometrien modellieren reale Objekte anhand von Koordinaten in einem n-dimensionalen Koordinatensystem und den Verbindungen zwischen diesen Punkten im virtuellen Raum. Die Verbindungen können durch verschiedene mathematische Funktionen modelliert werden, wobei Linien und sogenannte Splines, durch Funktionen höherer Ordnung modellierte kurvenförmige Verbindungen, am gebräuchlichsten sind.

Unterstützung für kurvenförmige, als auch 3-dimensionale Objekte entsprechend dem SQL/MM Standard ist zum Zeitpunkt der Erstellung in PostGIS integriert, jedoch ist diese Unterstützung noch in einem frühen Stadium und nur wenige der vorhandenen räumlichen Funktionen können diese Objekte korrekt verarbeiten.

Traditionell werden Geodaten hauptsächlich mithilfe von 2-dimensionalen Objekten und linearen Interpolationsmethoden modelliert, sodass die Unterstützung dieser Daten im Bezug auf räumliche Funktionen ausgereifter und umfangreicher ist. Auch andere GIS Applikationen wie z.B das hier verwendete gvSIG konzentrieren sich derzeit auf diese Datenarten, sodass auch die Import- und Exportfunktionen zu PostGIS noch keine volle Unterstützung für die neueren Geoobjekte bieten.

Ein Großteil der räumlichen Funktionen von PostGIS, soweit sie sich auf 2D Geometrien im kartesischen Raum beziehen, wird durch die GEOS Bibliothek, die ursprünglich eine C++ Portierung der Java Topology Suite (JTS) ist, zur Verfügung gestellt.

3.1 Datenstrukturen in PostGIS

3.1.1 Referenzsysteme in PostGIS

Grundlagen

Für die Darstellung von Koordinaten auf der Erdoberfläche werden in der Praxis grundsätzlich zwei Darstellungsformen verwendet. Dies sind winkelbasierte Koordinaten, die die Lage auf einem Ellipsoid anhand von Längen und Breitengraden angeben, und metrische Koordinaten, die die Lage auf einer durch Projektion verebneten Abbildung der Erdoberfläche angeben.

Abbildungen der gekrümmten Erdoberfläche auf die Ebene erzeugen prinzipbedingt Verzerrungen, die die räumlichen Daten dadurch zu einem bestimmten Maß verfälschen. Deshalb werden gebräuchliche Abbildungen wie UTM abschnittsweise für mehrere Teile bzw. Zonen der Erdoberfläche angewandt, sodass Verzerrungen in einem akzeptablen Rahmen gehalten werden.

Es gibt verschiedene Referenzellipsoide, die verwendet werden, um die Form der Erde näherungsweise zu beschreiben und verschiedene Projektionsmethoden, um diese Koordinaten zu verebnen. Die Kombination dieser und anderer Werte definieren das Referenzsystem der betroffenen Daten.

Umsetzung in PostGIS

Der von PostGIS umgesetzte Standard "Simple Features for SQL" der OGC stellt eine Tabelle „SPATIAL_REF_SYS“ zur Verfügung, in der eine große Zahl an Referenzsystemen definiert sind. Viele räumliche Funktionen verlangen den SRID des verwendeten Referenzsystems als Eingabewert.

Für die Reprojektion von Objekten in unterschiedliche Referenzsysteme greift PostGIS auf die Proj4 Bibliothek zurück.

Die SPATIAL_REF_SYS enthält folgende Felder:

- SRID – Der SRID ist eine Ganzzahl, die ein Referenzsystem in der SPATIAL_REF_SYS eindeutig bezeichnet und stellt den Primärschlüssel der Tabelle dar;
- auth_name – Name des Standards bzw. der Organisation der von der das entsprechende Referenzsystem übernommen wurde z.B EPSG;
- auth_srid – Die eindeutige Identifikationsnummer, wie sie von der Organisation, die unter auth_name bezeichnet wird, vergeben wurde;
- srtext – OpenGIS konforme Well Known Text Darstellung des Referenzsystems;
- proj4text – Die Proj4 Definition des Referenzsystems. Dieses Feld stellt zusammen mit auth_srid (bei bekanntem EPSG Code) eine der praktischsten Möglichkeiten zur Suche nach dem SRID für ein bestimmtes Referenzsystem dar.

Ein Beispiel für die Abfrage der SPATIAL_REF_SYS Tabelle nach dem im Tutorial verwendeten Referenzsystem, ETRS89 mit UTM Abbildung in der Zone 33, kann wie folgt aussehen:

```
SELECT
  spatial_ref_sys.srid,
  spatial_ref_sys.auth_name,
  spatial_ref_sys.auth_srid,
  spatial_ref_sys.proj4text
FROM
  public.spatial_ref_sys
WHERE auth_name='EPSG' AND auth_srid=25833;
```

mit folgendem Ergebnis:

srid	auth_name	auth_srid	proj4text
25833	EPSG	25833	+proj=utm +zone=33 +ellps=GRS80 +units=m +no_defs

oder wenn die EPSG Nummer unbekannt ist, anhand des `proj4text` Feldes:

```
SELECT
  spatial_ref_sys.srid,
  spatial_ref_sys.auth_name,
  spatial_ref_sys.auth_srid,
  spatial_ref_sys.proj4text
FROM
  public.spatial_ref_sys
WHERE
  proj4text LIKE '%ellps=GRS80%'
  AND proj4text LIKE '%proj=utm%'
  AND proj4text LIKE '%zone=33%'
  AND proj4text LIKE '%units=m%';
```

Das Ergebnis hier ist allerdings nicht eindeutig, sodass das richtige immer noch manuell ausgesucht werden muss:

srid	auth_name	auth_srid	proj4text
3006	EPSG	3006	+proj=utm +zone=33 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
3045	EPSG	3045	+proj=utm +zone=33 +ellps=GRS80 +units=m +no_defs
3767	EPSG	3767	+proj=utm +zone=33 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
4061	EPSG	4061	+proj=utm +zone=33 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
25833	EPSG	25833	+proj=utm +zone=33 +ellps=GRS80 +units=m +no_defs

Die wichtigsten Funktionen im Zusammenhang mit Referenzsystemen sind:

ST_SRID — gibt den SRID einer Geometrie zurück;

ST_SetSRID — setzt einen neuen SRID für eine Geometrie;

ST_Transform — erzeugt eine neue Geometrie, die in das durch einen neuen SRID angegebene Referenzsystem transformiert wurde.

Der SRID für unbekannte Referenzsysteme ist -1, sodass alle Funktionen und Datentypen, die einen SRID benötigen, auch bei unbekanntem Referenzsystem funktionieren. Es ist beabsichtigt, diesen Standardwert in Zukunft zu verändern. Der neue Wert für unbekannte Referenzsysteme soll aus Gründen der Standardkonformität 0 sein [vgl.06 S.36]

3.1.2 Die Geometry und Geography Datentypen

PostGIS bietet zwei unterschiedliche Datentypen für die Beschreibung von Vektordaten. Die beiden Typen unterscheiden sich in der Art der Koordinaten, die sie repräsentieren, und den dahinterliegenden mathematischen Modellen zur Verarbeitung dieser Daten.

Der Geometry Typ

Die Koordinaten des Geometry Typs stellen Punkte in einem kartesischen Koordinatensystem dar. Folglich beziehen sich Geometrien dieses Typs üblicherweise auf verebnete Abbildungssysteme bzw. Kartenprojektionen.

Vorteile der Verwendung des Geometry Typs:

- Die Berechnungen zugrunde liegende kartesische Mathematik ist relativ einfach und führt zu besserer Performance räumlicher Funktionen als beim Geography Typ.
- Der Geometry Typ wird seit Beginn der PostGIS Entwicklung unterstützt und Funktionen, die mit Geometrie Typ Daten arbeiten, können auf die GEOS Bibliothek zurückgreifen. Dies hat zur Folge, dass derzeit deutlich mehr Funktionen für Daten dieses Typs zur Verfügung stehen, als für den Geography Typ.
- Viele amtliche Geodaten liegen in festgelegten regionalen Projektionen wie z.B. UTM Zone 33N für Mecklenburg-Vorpommern vor, sodass im regionalen Umfeld auch auf diese Bezugssysteme zurückgegriffen werden sollte.

Nachteile der Verwendung des Geometry Typs:

- Die Berechnungen in verebneten Bezugssystemen werden grundsätzlich durch die abbildungsinhärenten Verzerrungen verfälscht. Bei sehr großen Gebieten sind die Ergebnisse stark verfälscht und ggf. gänzlich unbrauchbar.

- Berechnungen, die bestimmte Gebiete der Erde wie die Datumsgrenze oder die Pole betreffen bzw. berücksichtigen müssen, sind unter Verwendung des Geometry Typs meist verfälscht.
- Da sich alle Daten auf Projektionen, beziehen sind Kenntnisse über Kartenprojektionen nötig, insbesondere, wenn mehrere Gebiete mit verschiedenen Systemen zu verwalten sind.

Der Geography Typ

Die Koordinaten des Geography Typs stellen Längen- und Breitengrade auf einem Referenzellipsoid dar. Derzeit ist das einzige unterstützte Referenzsystem das WGS 84 bzw. EPSG 4326.

Vorteile der Verwendung des Geography Typs:

- Berechnungen über beliebig große Gebiete können korrekt ausgeführt werden.
- Räumliche Beziehungen, die die Pole oder die Datumsgrenze überschreiten, werden korrekt modelliert und berechnet.
- Das WGS 84 ist eines der am weitesten verbreiteten Bezugssysteme und gut geeignet, wenn Daten eines sehr großen Gebietes einheitlich verwaltet werden sollen.

Nachteile der Verwendung des Geography Typs:

- Der Ressourcenbedarf von Berechnungen ist aufgrund der aufwendigeren Mathematik grundsätzlich größer als beim Geometry Typ.
- Es stehen derzeit weniger räumliche Funktionen zur Verfügung.

Schlussfolgerung

Welcher Datentyp für die Speicherung von Datenbeständen empfehlenswerter ist, muss immer im Einzelfall entschieden werden. Wichtige Faktoren sind die räumliche Ausdehnung der Geoobjekte und das Bezugssystem der zur Verfügung stehenden Daten.

Es ist relativ einfach möglich die beiden Datentypen ineinander zu konvertieren, sodass die Vorteile beider kombiniert werden können.

Einschränkungen und Besonderheiten bei der Verwendung des Geography Typs

- Derzeit wird nur das WGS84 lon/lat (entspricht dem SRID4326) als Referenzsystem unterstützt.
- Funktionen, die Längen oder Flächen berechnen, geben das Ergebnis immer in Metern bzw. Quadratmetern an.
- Geography und Geometry Typ können ineinander konvertiert werden.
- Standardmäßig werden Geometry Typ Daten im WGS84 lon/lat (SRID4326) System in den Geography Typ konvertiert.
- Die Angabe der Koordinaten erfolgt nach dem Muster Länge/Breite nicht Breite/Länge.
- Es gibt noch keine Unterstützung für Kurvenobjekte.
- Optional können Funktionen ihre Berechnungen auf Basis eines kugelförmigen Erdmodells erstellen. Dies kann Geschwindigkeitsvorteile mit sich bringen, führt aber je nach Sachverhalt zu unterschiedlich großen Fehlern in den Ergebniswerten.
- Berechnungen legen immer die kürzeste Verbindung zweier Punkte auf dem Großkreis zugrunde, wodurch Formen, die Bögen von mehr als 180° enthalten, nicht korrekt modelliert und verarbeitet werden können.

- Berechnungen auf Basis des Geography Typs sind rechenintensiver als ebene Geometrieberechnungen. Insbesondere bei großen räumlichen Objekten ist es aus Geschwindigkeitsgründen oft notwendig, die Tabellen aufzuteilen, damit der räumliche Index besser genutzt werden kann und damit die Menge der zu ladenden und ggf. zu verarbeitenden Daten im Rahmen der Kapazitäten bleibt.
- Viele Funktionen verarbeiten Daten im Geography Typ, indem sie vorher nach Geometry konvertieren, die Berechnungen durchführen und anschließend zurück konvertieren. Dies erweitert einerseits die Funktionalitäten für diesen Typ, kann aber je nach Sachverhalt projektions- und konvertierungsbedingte Fehler herbeiführen.

3.1.3 Koordinatendimensionen

Die Koordinatendimension beschreibt die Dimension des Raumes, in dem sich ein Geoobjekt aufhält. Sie bestimmt, wie viele Zahlenwerte nötig sind um eine Koordinate zu beschreiben. Im Folgenden wird deshalb von Koordinatensets gesprochen.

PostGIS unterstützt bis zu 4-dimensioale Koordinaten, wobei der Koordinatenraum mindestens 2-dimensional ist.

Bei 3-dimensionalen Werten wird zwischen sogenannten „3DM“ und „3DZ“ Koordinaten unterschieden. Ist die dritte Dimension ein „Z“ Index, wird sie als räumliche Dimension behandelt, ist sie ein „M“ Wert, wird sie nicht als räumliche Dimension, sondern als Maßzahl (M steht für measure) verarbeitet. 4-dimensionale Koordinaten enthalten sowohl einen „Z“ als auch einen „M“ Wert.

Die Unterstützung für echte 3D Objekte in PostGIS ist noch relativ jung und nur ein Teil der Funktionen kann die zusätzliche Dimension verarbeiten. Alle Funktionen, die auf GEOS angewiesen sind, können nur den 2D Teil der Koordinaten verarbeiten.

Die Maßzahl-Dimension wird in einer Reihe von speziellen Funktionen verwendet und findet beim sogenannten „Linear Referencing“ Anwendung.

Die geometrische Dimension des Objektes kann kleiner, gleich aber niemals größer als die Koordinatendimension sein. [vgl.06 S. 35]

3.1.4 Geometrietypen

PostGIS stellt eine Reihe von Geometrietypen zur Verfügung, die verwendet werden können, um räumliche Sachverhalte zu modellieren. Grundsätzlich stehen alle Geometrietypen sowohl für Koordinaten im Geometry-, als auch im Geography Datentyp zur Verfügung.

In Bezug auf die Koordinatendimension der einzelnen Geometrietypen gibt es im Zuge von Veränderungen in der Metadatenverwaltung Unterschiede zu älteren Versionen von PostGIS. So gibt es für jeden Geometrietyp Untertypen, die sich durch ein angefügtes Z und/oder M kennzeichnen. Der Typ muss bei der Definition einer Tabelle angegeben werden.

Am Beispiel der Deklaration eines Punktes:

- 2D Objekt – POINT(0 0)
- 3DZ Objekt – POINTZ(0 0 0)
- 3DM Objekt – POINTM(0 0 0)
- 4D Objekt – POINTZM(0 0 0 0)

In älteren Versionen konnten die Geometrietypen jede Art von Koordinatendimension aufnehmen, nur 3DM Koordinaten mussten eigens deklariert werden. In diesem System wurden zur Erhaltung der Einheitlichen Dimension der Werte einer Spalte SQL CONSTRAINTs eingesetzt.[vgl.06 S.44]

In der folgenden Vorstellungen der Geometrietypen wird der Einfachheit halber meist von 2-dimensionalen Koordinatendimensionen ausgegangen.

Traditionelle Geometrietypen / OGC Standard

Diese Objekte sind Bestandteil des OGC SFS Standards, wobei der Standard selbst auf die 2D Koordinatendimension beschränkt ist. Sie werden in PostGIS seit längerem unterstützt. Gleichzeitig wird der Großteil dieser Typen traditionell für GIS Anwendungen genutzt und in der Regel von GIS Software gut unterstützt.

- POINT Objekte bzw. Punkte sind 0-dimensionale Objekte, die durch ein Koordinatenset definiert werden können.

➔ Beispiel: POINT(1 1)

- MULTIPOINT Objekte sind Sammlungen von Punkten, die nach außen wie ein einzelnes Punktobjekt angesprochen werden können.

➔ Beispiel: MULTIPOINT(1 1, 5 5, 9 3, 2 1)

- LINESTRING Objekte bzw. Linienketten sind 1-dimensionale Objekte und bestehen aus aneinander gereihten Liniensegmenten. Sie werden durch eine Abfolge von zwei oder mehr Koordinatensets definiert, durch welche die Linienkette verläuft. Sind der Anfangs- und Endpunkt einer Linienkette gleich, spricht man von einem geschlossenen Ring.

➔ Beispiel:

LINESTRING(1 1, 1 2, 3 6, 8 12, 10 11)	normaler LINESTRING
LINESTRING(0 0, 4 0, 4 4, 0 4, 0 0)	geschlossener Ring

- MULTILINESTRING Objekte sind Sammlungen von Linienketten, die nach außen wie ein einzelnes Linienkettenobjekt angesprochen werden können.

➔ Beispiel: MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))

- POLYGON Objekte bzw. Polygone sind 2-dimensionale Objekte. Sie bestehen aus einem äußeren Ring, der die Gesamtform des Polygons darstellt, und beliebig vielen inneren Ringen, die Löcher im Polygon definieren.

➔ Beispiel:

POLYGON((0 0,4 0,4 4,0 4,0 0))	Polygon ohne Loch
POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1))	Polygon mit Loch

- MULTIPOLYGON Objekte sind Sammlungen von Polygonen, die nach außen wie ein einzelnes Polygonobjekt angesprochen werden können.

➔ Beispiel:

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)))

Das Beispiel stellt ein Multipolygon dar, das aus zwei einzelnen Polygonen besteht, wobei das erste einen inneren Ring besitzt.

- GEOMETRYCOLLECTION Objekte sind spezielle Objekte, die eine beliebige Menge aller von PostGIS unterstützten Geometrietypen aufnehmen können. Sie ermöglichen die Speicherung heterogener Geometrien in einer Geometriespalte, ohne Konflikte mit den Metadaten zu verursachen. GEOMETRYCOLLECTIONs sind weniger als Geometrietyp für die Speicherung von Daten zu empfehlen, sind aber nötig, da bestimmte räumliche Abfragen heterogene Geometrien als Ergebnismenge liefern können, die dann in diesem Geometrietyp ausgedrückt werden können.[vgl. 06 S.45-46]

➔ Beispiel:

GEOMETRYCOLLECTION(POINT(2 3),LINESTRING(2 3,3 4),MULTIPOINT(1 1, 5 5, 9 3, 2 1))

Gekrümmte Geometrien

Im Zusammenhang mit dem SQL-MM Standard werden verschiedene Geometrietypen definiert, die auf gekrümmt interpolierten Bögen basieren. Die Unterstützung dieser Typen inklusive der im Standard geforderten Funktionen ist noch nicht abgeschlossen, soll aber in der finalen Version von Postgis-2.0 vollständig umgesetzt sein. Unterstützung für komplexere Interpolationen wie Splines und Bezier Kurven ist derzeit nicht vorhanden und stellt bis zur vollständigen Umsetzung der Anforderungen von SQL-MM für die einfacheren Objekte auch kein Entwicklungsziel dar. [vgl.06 S. 48-49]

- CIRCULARSTRING Objekte, oder in etwa übersetzt Kurvenketten, sind wie Linienketten 1-dimensionale Objekte. Ein Bogen wird durch drei Koordinatensets definiert, wobei das erste und dritte Set Anfangs- und Endpunkt des Bogens darstellen und das zweite Set ein beliebiger Punkt auf dem Bogen ist. Bei CIRCULARSTRINGs, die aus mehr als einem Bogen bestehen, stellt der Endpunkt des vorigen Bogens den Anfangspunkt des nächsten dar, sodass die Anzahl an Koordinatensets immer ungerade ist. Eine Sonderform dieses Typs sind geschlossene Kreise. Für Kreise sind Anfangs- und Endpunkt gleich, während der zweite Punkt auf der Mitte des Bogens also genau gegenüber des Startpunktes liegen muss.

➔ Beispiel:

CIRCULARSTRING(0 0, 1 1, 1 0)	einfacher Bogen
CIRCULARSTRING(0 0, 1 1, 1 0, 3 0, 3 3)	Kette aus zwei Bögen
CIRCULARSTRING(0 0, 4 0, 4 4, 0 4, 0 0)	geschlossener Kreis

- COMPOUNDCURVE Objekte bzw. zusammengesetzte Kurven sind ebenfalls 1-dimensional. Sie können sowohl CIRCULARSTRINGs als auch LINESTRINGs enthalten, wobei der Endpunkt der vorigen Geometrie immer gleich des Anfangspunktes der Nachfolgenden sein muss.

➔ Beispiel: COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0 1))

- MULTICURVE Objekte sind Sammlungen, die CIRCULARSTRINGs, LINESTRINGs und COMPOUNDCURVEs enthalten können. Wie alle Multiobjekte können sie nach außen wie ein einziges Objekt angesprochen werden.

➔ Beispiel: MULTICURVE((0 0, 5 5),CIRCULARSTRING(4 0, 4 4, 8 4))

- CURVEPOLYGON Objekte sind wie Polygone 2-dimensionale Geometrien und ebenfalls aus einem äußeren und beliebig vielen inneren Ringen aufgebaut. Der Unterschied besteht darin, dass die Ringe eines CURVEPOLYGONs aus CIRCULARSTRINGs, LINESTRINGs und COMPOUNDCURVEs bestehen können.

➔ Beispiel:

CURVEPOLYGON(CIRCULARSTRING(0 0, 4 0, 4 4, 0 4, 0 0),(1 1, 3 3, 3 1, 1 1))

CURVEPOLYGON(COMPOUNDCURVE(CIRCULARSTRING(0 0,2 0, 2 1, 2 3, 4 3),(4 3, 4 5, 1 4, 0 0)), CIRCULARSTRING(1.7 1, 1.4 0.4, 1.6 0.4, 1.6 0.5, 1.7 1))

- MULTISURFACE Objekte sind Sammlungen von POLYGON und CURVEPOLYGON Objekten. Sie können wie alle Multiobjekte nach außen wie ein einziges Objekt angesprochen werden.

➔ Beispiel:

MULTISURFACE(CURVEPOLYGON(CIRCULARSTRING(0 0, 4 0, 4 4, 0 4, 0 0),(1 1, 3 3, 3 1, 1 1)),((10 10, 14 12, 11 10, 10 10),(11 11, 11.5 11, 11 11.5, 11 11)))

Geometrietypen für 3D Geometrien

Die Unterstützung von 3-dimensionalen Geometrien befindet sich in PostGIS, sowie im GIS Bereich allgemein noch in einer relativ frühen Entwicklungsphase.

Geometrien können im 3-dimensionalen Raum gelagert werden (3DZ), wobei bereits viele räumliche Funktionen und ein 3D Index (siehe 3.1.6 zu räumlichen Indizes) zur Verfügung stehen.

Im Bezug auf Geometrien gibt es die zwei Typen TIN (Triangulated Irregular Network) und POLYHEDRALSURFACE, die genutzt werden können, um 3-dimensionale Körper zu bilden.

- TRIANGLE Objekte sind spezielle Polygone und stehen im Zusammenhang mit den TINs. Sie stellen immer Dreiecke ohne Löcher dar und dürfen somit nur 4 Koordinatensets mit identischem Anfangs- und Endpunkt beinhalten.

➔ Beispiel: TRIANGLE ((0 0, 0 9, 9 0, 0 0))

- TIN Objekte stellen eine Oberfläche aus Dreiecken dar. Sie können im 2D sowie 3D Raum definiert werden. Im 3D Raum können sie bei entsprechender Form ein Volumen einschließen und somit feste Körper definieren.

➔ Beispiel: TIN(((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0)))

- POLYHEDRALSURFACE Objekte stellen eine Oberfläche aus Polygonen dar. Sie verhalten sich im hier betrachteten Zusammenhang analog zu den TINs.

➔ Beispiel:

POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))

Die endgültigen Möglichkeiten dieser Geometrietypen, die sie betreffenden Funktionen oder Regeln für Validität stehen noch nicht fest und sind Gegenstand der Entwicklung.[vgl. 08]

3.1.5 Die GEOMETRY_COLUMNS und GEOGRAPHY_COLUMNS Metadatentabellen bzw. -VIEWs

Die GEOMETRY_COLUMNS Tabelle beinhaltet Metadaten über die Spalten aller Tabellen der Datenbank, die die Geometrieobjekte enthalten.

Sie ist Bestandteil des OGC Standards „Simple Features for SQL“ (SFS).

Applikationen, die auf PostGIS zugreifen, wie z.B Desktop GIS-Software, bestimmen die geladenen Daten oft anhand der Einträge in der Metadatentabelle. [vgl.06, S.34]

In allen früheren Versionen von PostGIS, als auch in den älteren Varianten von Postgis-2.0SVN ist die GEOMETRY_COLUMNS noch als „echte“ Tabelle implementiert. In der im Tutorial verwendeten Version existiert die GEOMETRY_COLUMNS nicht mehr als Tabelle, sondern als VIEW, dessen Datenfelder dynamisch aus den Systemkatalogen generiert werden. Die Metadaten des wesentlich neueren Geography Typs wurden bereits vorher als entsprechender VIEW gespeichert. [vgl. 07]

Die Datenfelder des neuen GEOMETRY_COLUMNS VIEWs haben dieselben Bezeichnungen wie die der alten Tabelle. Es ist aus Gründen der Kompatibilität zu vermuten, das sich die neue Implementierung der Metadatenhaltung nach außen, also gegenüber Abfragen und insbesondere Drittanwendungen, weitgehend genauso verhält bzw. verhalten soll wie die bisherige GEOMETRY_COLUMNS Tabelle. Welche Veränderungen sich aus diesem Wechsel endgültig ergeben, steht zum Erstellungszeitpunkt des Tutorials noch nicht fest bzw. ist noch nicht aus der Dokumentation ersichtlich.

GEOMETRY_COLUMNS beinhaltet folgende Daten:

- f_table_catalog – Der Name des „catalog“ im Pfad zur Spalte. Dieses Feld enthält in PostGIS immer den Namen der Datenbank, da in PostgreSQL kein Gegenstück zu „catalogs“ existiert. Das Feld existiert aus Gründen der Standardkonformität.
- f_table_schema – Der Name des Schemas im Pfad zur Spalte. Schemas in PostgreSQL sind logische Gruppierungen für Tabellen. [vgl.05]
- f_table_name – Der Name der Tabelle im Pfad zur Spalte.
- f_geometry_column – Der Name der Geometriespalte
- coord_dimension – Die räumliche Dimension der Koordinaten der Geometrien der Spalte. Zu diesem Attribut existiert ein CONSTRAINT in ordnungsgemäß registrierten Geometriespalten.
- Srid – Der SRID zur Bestimmung des räumlichen Referenzsystems der Geometrien. Zu diesem Attribut existiert ein CONSTRAINT in ordnungsgemäß registrierten Geometriespalten.
- Type – Der Typ der gespeicherten Geometrien z.B. POLYGON oder MULTIPOINT. Zu diesem Attribut existiert ein CONSTRAINT in ordnungsgemäß registrierten Geometriespalten.

Für den Geography Typ existiert analog ein VIEW GEOGRAPHY_COLUMNS, der im Wesentlichen dieselben Informationen Spalten mit Geography Typ basierten Daten enthält. Lediglich das Feld „f_geometry_column“ heißt dort entsprechend „f_geography_column“.

Umgang mit der alten GEOMETRY_COLUMNS Tabelle

Die ältere GEOMETRY_COLUMNS Tabelle wird nicht automatisch mit dem tatsächlichen Datenbestand abgeglichen, sodass Inkonsistenzen auftreten können. Dies ist vor allem dann möglich, wenn Geometriespalten ohne die Funktion „AddGeometryColumn“ erstellt werden, im Nachhinein CONSTRAINTS verändert werden oder wenn die Metadatentabelle selbst editiert wird.

Um die Metadaten mit den normalen Tabellen konsistent zu halten, empfiehlt es sich wenn möglich für Aktionen, die die GEOMETRY_COLUMNS betreffen, auf die entsprechenden PostGIS Funktionen zurückzugreifen.

In PostGIS existieren folgende Funktionen zum Umgang mit der GEOMETRY_COLUMNS Tabelle: [Funktionsbeschreibungen vgl. 06 S.37]

- AddGeometryColumn – Diese Funktion legt eine Spalte für Geometriedaten an und registriert diese in der Metadatentabelle. Dabei werden drei CONSTRAINTs angelegt, die die Einhaltung der Festlegungen in den Feldern „coord_dimension“, „srid“ und „type“ sicherstellen.
- DropGeometryTable – löscht eine Tabelle, die eine Geometriespalte enthält, und die entsprechenden Metadaten.
- UpdateGeometrySRID – ändert den SRID aller Geometrien einer Tabelle und aktualisiert auch das „srid“ Feld der GEOMETRY_COLUMNS Tabelle entsprechend. Mit dieser Änderung ist ausdrücklich keine Reprojektion der Koordinaten verbunden. Diese Funktion kann eingesetzt werden, falls eine Tabelle mit einem falschen SRID angelegt wurde, beispielsweise weil ein GIS Programm das Referenzsystem einer Datei nicht erkannt und als unbekannt festgelegt hat.
- Probe_Geometry_Columns – durchsucht die Datenbank nach Tabellen mit relevanten CONSTRAINTs und fügt deren Metadaten der GEOMETRY_COLUMNS hinzu, wenn diese noch nicht registriert waren.
- Populate_Geometry_Columns – inspiziert alle Tabellen und VIEWs einer Datenbank, ermittelt für Tabellen mit Geometriespalten die notwendigen Informationen für die Einträge in die

Metadatentabelle anhand der Datensätze und erzeugt diese Metadaten anschließend. Im Fall von Tabellen werden, wenn noch nicht vorhanden, die drei CONSTRAINTs hinzugefügt, die auch von AddGeometryColumn erzeugt werden. Die Funktion kann also eingesetzt werden, wenn der Verdacht auf Inkonsistenzen mit den Metadaten besteht.

Umgang mit dem neuen GEOMETRY_COLUMNS VIEW

Ziel der neuen Implementierung ist unter anderem, die Konsistenz der Metadaten mit den Tabellen automatisch zu gewährleisten.[vgl. 07]

Durch diese Umstellung ist die Nutzung der o.g. Funktionen weitgehend unnötig geworden. Die Funktion „Probe_Geometry_Columns“ existiert in den neuen Versionen nicht mehr. Die anderen Funktionen sind weiterhin vorhanden und funktionieren weitgehend wie bisher. Dies hat den Vorteil, dass ältere Abfragen und SQL Skripte weiterhin funktionieren. Inwieweit es Änderungen im Verhalten dieser Funktion gibt, ist zum Erstellungszeitpunkt noch nicht dokumentiert. Eine Änderung der „AddGeometryColumn“ Funktion besteht darin, das sie nicht mehr die drei o.g. CONSTRAINTs erstellt, da diese durch das neue System überflüssig geworden sind.

Der Prozess der Erstellung neuer Tabellen mit Geometriespalten ändert sich gegenüber früheren Versionen ebenfalls. Auf diesen Punkt wird im entsprechenden Teil des Tutorials eingegangen.

3.1.6 Räumliche Indizes

Indizes in Datenbanksystemen sind Datenstrukturen, die Abfragen über indizierte Daten beschleunigen. Dies wird grundlegend dadurch erreicht, dass ein großer Teil der betroffenen Wertebereiche eines Feldes im Vorhinein ausgeschlossen werden kann und nicht auf Übereinstimmung mit dem eigentlichen Abfragekriterium überprüft werden muss.

PostgreSQL verwendet üblicherweise „b-tree“ Indizes, die Attributwerte entlang einer Skala, beispielsweise das Alphabet bei Wörtern, ordnen und dann Suchbäume erstellen. Diese Art Index steht auch bei PostGIS Tabellen zur Verfügung, ist aber für räumliche Attribute nicht sinnvoll einsetzbar. Die Verwendung normaler „b-tree“ Indizes auf nicht-räumliche Attribute einer PostGIS Tabelle kann sinnvoll sein, wenn diese Attribute entsprechend oft Gegenstand von Abfragen sind.

PostGIS stellt einen für räumliche Attribute einsetzbaren Index auf Basis des „GIST“ (Generalized Search Tree) Indextyps von PostgreSQL zur Verfügung.

Dieser Index arbeitet auf Basis der Bounding Boxen der Geoobjekte, sodass die garantiert nicht bedeutsamen Geometrien mithilfe relativ unaufwendiger Algorithmen ausgeschlossen werden können. Weiterhin führt dieses Verfahren zu einer Verkleinerung der Indizes, da zur Definition einer Bounding Box bekanntermaßen nur zwei Koordinatenpaare notwendig sind.

Für die Abfrage räumlicher Beziehungen anhand der Bounding Boxen sind in PostGIS eigens Operatoren vorhanden, die u.a. eingesetzt werden, um Abfragen durch Indexnutzung zu beschleunigen. Zusätzlich beinhalten viele der Funktionen zur Prüfung der räumlichen Beziehung zwischen Objekten implizit Abfragen bezüglich der Überlappung der Bounding Boxen. Dadurch kann oft auf den expliziten Einsatz von räumlichen Operatoren verzichtet werden.

Die Erstellung eines räumlichen Indizes läuft nach folgendem Muster ab:

```
CREATE INDEX Indexname ON tabelle USING GIST (geometriefeld);
```

Der Name des Indizes kann frei gewählt werden, enthält jedoch meist mindestens den Tabellennamen.

Grundsätzlich können Indizes lesende Abfragen erheblich beschleunigen, gleichzeitig müssen sie aber bei jeder Änderung an einem indizierten Datenfeld aktualisiert werden, was wiederum Ressourcen in Anspruch

nimmt. Prinzipiell sollten nur Felder indiziert werden, die oft lesend abgefragt, aber nicht konstant verändert oder ergänzt werden.

3.1.7 Validierung von Geometrien

Grundsätzlich können in PostGIS alle Geometrien gespeichert werden, solange sie die Anforderungen der entsprechenden Datentypen erfüllen. Viele der PostGIS Funktionen zur Verarbeitung von Geometrien setzen jedoch voraus, dass die übergebenen Objekte den OpenGIS Standard des OGC entsprechen.

Die Anforderungen zur Speicherung richten sich ausschließlich auf die korrekte Darstellbarkeit des Datensatzes im gewählten Geometrietyp. Beispiele für diese Art an Anforderungen sind:

- Koordinaten müssen je nach Koordinatendimension in Sets von 2, 3 oder 4 Zahlenwerten angegeben werden;
- Der erste und letzte Punkt eines Polygonrings müssen identisch sein;
- Objekte des Typs „Triangle“ dürfen nur vier Koordinatensets besitzen, wobei der erste und letzte Punkt identisch sein müssen.

Kriterien für gültige Geometrien

Die OpenGIS Standards des OGC verlangen, dass Geometrien einfach (simple) und gültig (valid) sein müssen.

Als einfach gelten Geometrien, die keine anomalen Punkte beinhalten. Insbesondere Selbstüberschneidung und Selbstberührung von einfachen Geometrien ist ausgeschlossen. Dieses Kriterium gilt hauptsächlich für punkt- und linienartige Objekte, da Polygone grundsätzlich einfach im Sinne dieser Definition sind.

Im Einzelnen bestehen die Bedingungen für Einfachheit für die einzelnen Geometrietypen aus folgenden Punkten:

- Punkte sind grundsätzlich immer einfach;
- Multipunktobjekte dürfen keine zwei Punkte mit gleichen Koordinaten enthalten;
- Linienobjekte dürfen keinen Punkt mit Ausnahme des Startpunktes zweimal durchlaufen;
- Multilinien müssen aus einfachen Linien bestehen. Die einzelnen Linien dürfen sich nicht überschneiden und nur mit ihren Anfangs- und Endpunkten berühren.

Die Gültigkeit bzw. Validität bezieht sich hauptsächlich auf Polygone und stellt folgende Bedingungen:

Polygone:

- Keiner der inneren Ringe eines Polygons darf ganz oder teilweise außerhalb des äußeren Ringes liegen;
- Die Ringe, die ein Polygon bilden, dürfen sich nicht überschneiden und nur an maximal einem Punkt berühren;
- Polygone dürfen keine linienartigen, also nicht-flächenhafte, Ein- oder Ausstülpungen besitzen.

Multipolygone:

- Multipolygone müssen aus gültigen Polygonen bestehen;
- Die einzelnen Polygone dürfen sich nicht überschneiden;
- Die äußeren Grenzen der einzelnen Polygone dürfen sich nur in einer endlichen Anzahl an Punkten berühren. Dies schließt beispielsweise direkt aneinander grenzende Polygone aus, deren Grenzen sich in einer Linie, also unbegrenzt vielen Punkten, berühren.

Linienketten:

- PostGIS definiert zusätzlich die Einschränkung, dass gültige Linienketten mindestens zwei verschiedene Punkte enthalten müssen.

Validierungsmöglichkeiten in PostGIS

Zur Überprüfung der Einfachheit und Gültigkeit von Geometrien stehen die Funktionen ST_IsSimple und ST_IsValid zur Verfügung. Diese Funktionen können auch als CONSTRAINT in einer Tabelle eingebunden werden, um dort nur entsprechend standardkonforme Geometrien zuzulassen. Dies wird nicht standardmäßig eingerichtet, da die Prüfung auf Einfachheit und Gültigkeit relativ viel Rechenzeit beansprucht.

Für genauere Betrachtung ungültiger Geometrien gibt es zwei weitere Funktionen:

- ST_IsValidReason, gibt einen Text mit der Gültigkeit und ggf. dem Grund der Ungültigkeit zurück.;
- ST_IsValidDetail, gibt ein spezielles Objekt mit der Gültigkeit sowie dem Grund und Ort der Ungültigkeit zurück.

Korrekturmöglichkeiten für ungültige Geometrien

Grundsätzlich ist es immer möglich und u.U. Empfehlenswert, die ungültigen Daten manuell zu überprüfen und zu berichtigen, wobei die Daten von ST_IsValidDetail nützlich sein können.

PostGIS bietet die Funktion ST_MakeValid an. Diese Funktion kann auf Polygone und Linienketten sowie deren Multi-Varianten angewandt werden und erstellt eine geänderte gültige Variante der Geometrie. Dabei wird versucht, die Koordinaten der Ursprungsgeometrie weitgehend zu erhalten. Der Geometrietyp kann sich dabei von Polygonen zu Multipolygonen verändern, um Selbstüberschneidungen zu vermeiden. Falls Teile der Ausgangsgeometrie zu einem Objekt mit geringerer räumlicher Dimension reduziert werden müssen, wird die neue Geometrie als Geometrycollection zurückgegeben.

Eine weitere Konvention für Geometrien besteht in der sogenannten „Recht-Hand-Regel“, d.h. der Anforderung, dass die Fläche eines Polygons rechts von seiner Begrenzung liegen muss. Diese Konvention betrifft die Ausrichtung der Ringe in Polygonen, wobei die Koordinaten des äußeren Ringes im Uhrzeigersinn und die der inneren Ringe entgegengesetzt definiert werden müssen. Diese Forderung ist nicht Teil der in PostGIS umgesetzten Standards und für die Arbeit mit PostGIS nicht notwendig. Es ist möglich, diese Art der Ausrichtung mithilfe der Funktion ST_ForceRHR für die übergebenen Geometrien herzustellen.

3.2 Speichern und Laden

3.2.1 OpenGIS Standard Formate

Die Speicherung und Abfrage von räumlichen Daten erfolgt grundsätzlich über SQL Befehle. Dabei stehen verschiedene Formate zur Verfügung, die im Folgenden beschrieben werden:

Der OpenGIS Standard definiert die sogenannten Well-known Text (WKT) und Well-known Binary (WKB) Formen, um räumliche Objekte auszudrücken. Diese enthalten in einem spezifischen Format die Art des Objektes und die dazugehörigen Koordinaten. Das WKT Format bietet dabei eine vom Menschen lesbare Form, ist in der Genauigkeit der Koordinaten jedoch eingeschränkter als WKB. Der Standard schreibt vor, dass das Referenzsystem, auf das sich die Koordinaten beziehen, intern gespeichert werden muss.

Die standardisierten WKB und WKT Formate unterstützen nur 2D Geometrien und erlauben es nicht, den SRID (Spatial Reference System Identifier) in das Format zu integrieren.

PostGIS definiert eigene erweiterte Formate, die sogenannten Extended Well Known Text (EWKT) und Extended Well Known Binary (EWKB) Formate. Diese Formate erweitern die Obigen um die Möglichkeit, 3D und 4D Koordinaten sowie zusätzliche Objekte wie z.B. Kurven und TINs zu nutzen. Zusätzlich kann die Repräsentation im EWK(T/B) Format Informationen zum SRID enthalten.

Eine Abwandlung des EWKB, das HEXEWB ist eine hexadezimale Darstellung der binären Form des EWKB. Es ist das Format, das in entsprechenden INSERT und SELECT Anweisungen ohne Aufruf zusätzlicher Funktionen das Eingabe- und Ausgabeformat von Geometrien darstellt. Das HEXEWKB wird deswegen auch als „canonical Form“ bezeichnet, was in diesem Zusammenhang als Normalform übersetzt werden kann.

Zur Demonstration dieser Formate soll ein einfaches Polygon aus der Tabelle „tutopoly“, die weiter unten angelegt wird, dienen:

```
SELECT
  gid,
  name,
  geom AS HEXEWKB,
  ST_AsText(geom) AS WKT,
  ST_AsEWKT(geom) AS EWKT,
  ST_AsBinary(geom) AS WKB,
  ST_AsEWKB(geom) AS EWKB
FROM public.tutopoly
WHERE gid=1;
```

SQL.Ausgabe:

gid	name	hexewkb	wkt	ewkt
2	Testdreieck	0103000020E96400	POLYGON((1 1,2.5 5,5 1,1 1))	SRID=25833;POLYGON((1 1,2.5 5,5 1,1 1))

wkb	ewkb
)0\000\000\000\000\004@\000\000\000\000\000\000\024@\	00\000\000\000\000\000\004@\000\000\000\000\000\000\0

Hinweis: die Repräsentation der Daten in HEXEWKB, WKB und EWKB ist ausgeschrieben sehr lang und wurde aus Platzgründen nur ausschnittsweise dargestellt.

3.2.2 Erstellung von Tabellen und Laden räumlicher Daten mittels SQL

In diesem Abschnitt soll die Beispieltabelle „tutopoly“ erstellt und mit einfachen Datensätzen gefüllt werden.

Die hier dargestellten Schritte können manuell so ausgeführt werden, ihr primärer Zweck liegt jedoch in der Darstellung der grundlegenden Bestandteile einer PostGIS Tabelle und der beteiligten Anweisungen zur Erstellung.

Aufgrund der Flexibilität der SQL-Konstrukte können die dargestellten Schritte auch anders, und möglicherweise effizienter, ausgedrückt werden.

Zunächst wird der Prozess für die älteren PostGIS Versionen, die noch mit der GEOMETRY_COLUMNS Metadatentabelle arbeiten, dargestellt, da der Großteil der bestehenden PostGIS Datenbanken noch auf dem alten System basiert. Anschließend wird auf die Änderungen durch den neuen GEOMETRY_COLUMNS VIEW eingegangen.

Leistungsbetrachtungen wurden hier außer Acht gelassen, sind aber z. B. für Client-Applikationen, die automatisiert ggf. sehr große SQL-Skripte erzeugen, von Bedeutung.

Vorgehen bei älteren PostGIS Versionen mit einer GEOMETRY_COLUMNS Tabelle

Erstellung einer PostgreSQL Tabelle

Zunächst wird eine einfache Tabelle erzeugt. Diese Tabelle kann für ihre Attribute alle Datentypen nutzen, die PostgreSQL anbietet. Tutopoly soll mit zwei nicht geometrischen Attributen erstellt werden, einem automatisch inkrementierbaren Zahlenwert, der als Primärschlüssel geeignet ist, und einem Textfeld zur Benennung.

```
CREATE TABLE Tutopoly(
      gid SERIAL,
      name text
);
```

Ergebnismeldung:

```
HINWEIS:  CREATE TABLE erstellt implizit eine Sequenz »testor_gid_seq« für die
»serial«-Spalte »testor.gid«
Abfrage war erfolgreich nach 203 ms. Keine Zeilen geliefert.
```

Die Tabelle wurde also erfolgreich erstellt. Die Erstellung der Sequenz, ausgelöst durch das SERIAL Attribut, ermöglicht die spätere Verwendung von „gid“ als automatisch inkrementierendes Feld.

Erstellung und Registrierung der Geometrie Spalte

Üblicherweise wird die Spalte für PostGIS Datenobjekte des Geometrie Typs durch die Funktion „AddGeometryColumn“ angelegt. Dies hat die Registrierung der neuen Spalte in der Metadaten Tabelle „GEOMETRY_COLUMNS“ und die Erstellung von drei CONSTRAINTS, worauf im nächsten Schritt eingegangen wird, zur Folge.

```
SELECT AddGeometryColumn('tutopoly',
'geom', 25833, 'POLYGON',2);
```

Ergebnis:

addgeometrycolumn
public.tutopoly.geom SRID:25833 TYPE:POLYGON DIMS:2

Es ist ebenfalls möglich, die neue Tabelle über eine normale INSERT Anweisung in der Metadatentabelle zu registrieren. Dies kann notwendig sein, z.B. wenn VIEWS erstellt werden sollen. Dabei werden natürlich auch keine CONSTRAINTS erstellt, sodass diese wenn gewünscht ebenfalls manuell erzeugt werden müssten. In diesem Fall müssen die Eintragungen in die Metadatentabelle manuell erstellt werden.

Erstellen von CONSTRAINTS

CHECK CONSTRAINTS in PostgreSQL sind Regeln, die bei jedem Einfügen neuer Datensätze überprüft werden und verhindern, dass Daten eingetragen werden, die nicht dem beabsichtigten Datenmodell entsprechen.

Die Beispieltabelle soll vier CONSTRAINTS enthalten, die auch von dem später behandelten "shp2pgsql" Importprogramm erzeugt werden. Die letzten drei wurden bereits durch AddGeometryColumn erzeugt, sodass nur der erste CONSTRAINT erzeugt werden muss. Dennoch wird hier zur Veranschaulichung die SQL Anweisung zu Erstellung aller CONSTRAINTs gezeigt, wie sie bei einer manuell erstellten Tabelle nötig wäre.

```
ALTER TABLE tutopoly
      ADD CONSTRAINT tutopoly_pkey PRIMARY KEY(gid),
      ADD CONSTRAINT enforce_dims_geom CHECK (st_ndims(geom) = 2),
      ADD CONSTRAINT enforce_geotype_geom CHECK (geometrytype(geom) =
      'POLYGON'::text OR geom IS NULL),
      ADD CONSTRAINT enforce_srid_geom CHECK (st_srid(geom) = (25833));
```

- Der CONSTRAINT "tutopoly_pkey" macht das Feld "gid" zum Primärschlüssel der Tabelle.
- Über "enforce_dims_geom" wird jede neue Geometrie durch die "st_ndims()" Funktion geprüft, die die räumliche Dimension der Koordinaten der Geometrie ermittelt. Der CONSTRAINT erzwingt also, das nur Körper im 2-Dimensionalen Raum eingefügt werden dürfen.
- Der CONSTRAINT „enforce_geotype_geom“ stellt sicher, dass nur Geometrien einer bestimmten Art, hier Polygone, in die Tabelle geladen werden können. Andere Geometrien wie Liniendaten oder Punkte werden abgelehnt, durch den Ausdruck „OR geom IS NULL“ ist es aber möglich, dass Datensätze ohne Geometrien erzeugt werden können.
- Der vierte CONSTRAINT „enforce_srid_geom“ prüft über die Funktion st_srid das verwendete Referenzsystem und lässt nur Datensätze des Systems mit dem SRID 25833 (ETRS89, UTM Zone 33N) zu.

Grundsätzlich verwenden PostgreSQL CONSTRAINTs normale Ausdrücke und Funktionen, die zumindest im Fall der PostGIS Funktionen auch in anderen Abfragen verwendet werden können.

Daraus folgt, dass je nach Sachverhalt unterschiedliche CONSTRAINTs für die Geometrien und anderen Attribute definiert werden können, um die Einhaltung der jeweiligen Anforderung an die Daten sicherzustellen.

Gleichzeitig sind CONSTRAINTs auch für räumliche Tabellen optional, sodass es beispielsweise möglich ist, Tabellen zu definieren, die Datensätze mit beliebigen Referenzsystemen zulässt, die erst später transformiert werden sollen.

Eine für die Konzeptionierung einer neuen PostGIS Datenbank wichtige Entscheidung ist in diesem Zusammenhang die Frage, ob Tabellen wie im Beispiel auf einen Geometrietyp beschränkt werden sollten oder nicht. Spalten mit heterogenen Geometrien können in bestimmten Fällen Abfragen vereinfachen, insbesondere wenn die anderen Attribute dieser Tabelle für die Objekte mit unterschiedlichen Geometrien gleich sind. Im Interesse der verlässlichen Funktion räumlicher Abfragen, die bestimmte Geometrietypen voraussetzen, und der allgemeinen Konsistenz der Daten ist üblicherweise der obige Ansatz mit nur einem Geometrietyp pro Spalte vorzuziehen. Eine weitere relativ komplexe Möglichkeit zur Strukturierung stellt die Nutzung der Tabellen Vererbung von PostgreSQL dar, die es ermöglicht, ähnlich der Vererbung und

Polymorphie in der Objektorientierten Programmierung abstrakte Elterntabellen einzurichten, deren Kindtabellen jeweils unterschiedliche Geometrietypen enthalten, die aber durch Abfragen gegen die Elterntabelle wie eine einzige Tabelle abgefragt werden können.[vgl.06 S.53-60]

Vorgehen bei neuen PostGIS Versionen mit einem GEOMETRY_COLUMNS VIEW und für Daten des Geography Typs

Das neue System zur Metadatenverwaltung ist darauf ausgelegt, neue Tabellen automatisch zu registrieren und die Eigenschaften SRID, Koordinatendimension und Geometrietyp der Geometriespalten ebenfalls automatisch zu überprüfen.

Dies hat zur Folge, dass das räumliche Attribut zusammen mit den anderen in der CREATE Anweisung definiert werden kann.

Tabellenerstellung

```
CREATE TABLE tutopoly (
      gid SERIAL,
      name text,
      geom GEOMETRY(POLYGON,25833)
);
```

Erstellen von CONSTRAINTS

Aufgrund des neuen Metadatensystems müssen die drei CONSTRAINTs zur Erhaltung der Homogenität der Tabellen nicht mehr erstellt werden.

Der einzige normalerweise angelegte CONSTRAINT dient der Herstellung des Primärschlüssels

```
ALTER TABLE tutopoly
      ADD CONSTRAINT tutopoly_pkey PRIMARY KEY(gid),
```

Es ist natürlich weiterhin möglich, andere eigene CONSTRAINTs zu definieren, um beispielsweise nur validierte Geometrien zuzulassen.

Die Tabellenerstellung für den Geography Typ verhält sich analog dazu. Tutopoly würde hier wie folgt erstellt werden:

```
CREATE TABLE geog_tutopoly (
      gid SERIAL,
      name text,
      geog GEOGRAPHY(POLYGON,4326)
);
```

Einfügen von Datensätzen

Von diesem Punkt an unterscheidet sich die Vorgehensweise zwischen den älteren und neuen PostGIS Versionen nicht mehr.

In diesem Schritt werden testweise die Daten eines Dreiecks in die Tabelle eingefügt. Für die Speicherung der Geometrie wird die Funktion „ST_GeomFromEWKT()“ eingesetzt, die eine Geometriebeschreibung nach dem EWKT Format entgegennimmt und in die Datenbank speichert.

Äquivalente Funktionen existieren für EWKB, WKB und WKT, wobei der Funktionsname hier „ST_GeometryFromText()“ lautet.

Zusätzlich ist zu beachten, dass die Eingabefunktionen für WKT und WKB zwingend die Angabe eines SRID benötigen, da dieser nicht in den beiden Formaten enthalten ist, aber laut Open GIS Standards in der Datenbank gespeichert werden muss.

Geometrien im HEXEWKB Format können ohne Eingabefunktion direkt eingefügt werden.

```
INSERT INTO tutopoly (gid,name,geom) VALUES
(1,'Testdreieck',ST_GeomFromEWKT('SRID=25833;POLYGON(( 1 1,2.5 5,5 1,1 1))'))
```

Analog dazu existieren für den Geography Typ die Funktionen ST_GeogFromText und ST_GeogFromWKB,

Die INSERT Anweisung für geog_tutopoly sieht wie folgt aus:

```
INSERT INTO geog_tutopoly (gid,name,geog) VALUES (1,'Testpolygon',
ST_GeographyFromText('SRID=4326;POLYGON(( 1 1,2.5 5,5 1,1 1))'));
```

Ergebnismiteilung:

```
Abfrage war erfolgreich durchgeführt: 1 Zeile, 62 ms Ausführungszeit.
```

3.2.3 Zusätzliche Ein- und Ausgabeformate

Neben den Standardformaten des OpenGIS Standards unterstützt PostGIS die Speicherung und Ausgabe von Geometrien in verschiedenen anderen Formaten. Dies erfolgt nach demselben Muster wie bei den WK(BT) Formaten.

Zur Speicherung stehen folgende Funktionen zur Verfügung:

- ST_GeomFromGML – erzeugt eine PostGIS Geometrie aus einer textuellen Geometriebeschreibung in OGC GML Syntax. Dabei dürfen nur die geometriebezogenen Teile eines GML Dokuments verwendet werden.
- ST_GeomFromKML – erzeugt eine PostGIS Geometrie aus einer textuellen Geometriebeschreibung in OGC KML Syntax. Dabei dürfen nur die geometriebezogenen Teile eines KML Dokuments verwendet werden.

Im Folgenden werden die zusätzlichen Ausgabefunktionen zusammen mit je einem Beispiel aufgelistet.

- ST_AsGML – gibt das entsprechende PostGIS Geometrieobjekt als Text in GML Syntax aus. Dabei kann zwischen GML Version 2 und 3 gewählt werden.

➔ `SELECT AsGML(geom) FROM tutopoly WHERE name='Testdreieck'`

gml
<gml:Polygon srsName="EPSG:25833"><gml:outerBoundaryIs><gml:LinearRing><gml:coordinates>1,1 2.5,5 5,1 1,1</gml:coordinates></gml:LinearRing></gml:outerBoundaryIs></gml:Polygon>

- ST_AsKML – gibt das entsprechende PostGIS Geometrieobjekt als Text in KML Syntax aus.

➔ `SELECT AsKML(geom) FROM tutopoly WHERE name='Testdreieck'`

kml
<Polygon><outerBoundaryIs><LinearRing><coordinates>10.51126507428976,0.000009019375921 10.511278512783747,0.000045096880442 10.511300910277825,0.000009019376367 10.51126507428976,0.000009019375921</coordinates></LinearRing></outerBoundaryIs></Polygon>

- ST_AsGeoJSON – gibt das PostGIS Geometrieobjekt als GoJSON Objekt aus

➔ `SELECT AsGeoJSON(geom) FROM tutopoly WHERE name='Testdreieck'`

geojson
{"type":"Polygon","coordinates":[[[1,1],[2.5,5],[5,1],[1,1]]]}

- `ST_AsSVG` – gibt das entsprechende PostGIS Geometrieobjekt als SVG Pfad aus

➔ `SELECT AsSVG(geom) FROM tutopoly WHERE name='Testdreieck'`

svg
M 1 -1 L 2.5 -5 5 -1 Z

- ST_GeoHash – gibt eine Repräsentation der Geometrie im GeoHash Format zurück. Die Funktion verarbeitet nur Objekte mit Längen- und Breitenkoordinaten, aber keine Objekte vom Geography Typ. Geohash ist dafür ausgelegt, Punktgeometrien zu beschreiben.

➔ `SELECT ST_GeoHash('POINT (-3.2342342 -2.32498)') AS GeoHash`

geohash
7ztdy2ubhbk4n42xg760

- ST_AsLatLonText – gibt eine Beschreibung des Geometrieobjekts als Längen- und Breitengrade, Winkelminuten und Winkelsekunden zurück. Diese Ausgabefunktion verarbeitet nur Punktgeometrien.

➔ `SELECT ST_AsLatLonText('POINT (-3.2342342 -2.32498)') AS latlong`

latlong
2Â°19'29.928"S 3Â°14'3.243"W

- ST_AsX3D – gibt eine Repräsentation der Geometrie als „X3D xml node element“ zurück. Die Funktion befindet sich noch in einer unfertigen Entwicklungsphase, sodass besonders 2D Objekte noch nicht oder nicht korrekt wiedergegeben werden.

3.2.4 Konvertierung von ESRI Shapefiles

Shapefiles stellen für geographische Daten einen de facto Standard als Datenaustauschformat dar. Es ist daher davon auszugehen, dass ein beträchtlicher Teil der Daten, die in einer PostGIS Datenbank gespeichert werden sollen, in diesem Format vorliegen oder irgendwann als Shape Datei exportiert werden müssen.

Für diesen Zweck stellt PostGIS zwei Kommandozeilen Programme zur Verfügung, "shp2pgsql" zum Laden von Shapefiles in die Datenbank und "pgsql2shp" zum Exportieren von PostGIS Daten in das Format.

Nutzung des Shapefile Imports durch shp2pgsql

Der grundlegende Aufruf von shp2pgsql erfolgt nach folgendem Muster:

```
shp2pgsql [Optionen] Name_des Shapefiles Name_der_Zieltabelle
```

Die Zieltabelle wird in der Notation „Schemaname.Tabellenname“ angegeben. Wird kein Schema angegeben, nimmt das Programm an, dass die Tabelle im Schema „public“ liegt.

Shp2pgsql stellt verschiedene Optionen in Form von Parametern bereit von denen einige in Folgenden kurz aufgeführt werden sollen:

- -c

➔ Erzeugt eine neue Tabelle und füllt sie mit den Daten des Shapefiles. Dies ist die Standardeinstellung, wenn keiner der vier Parameter -c,-a,-d oder -p gegeben ist

- -a

➔ Fügt die geladenen Daten zu denen einer vorhandenen Tabelle hinzu. Dies setzt voraus, dass alle Datensätze dieselben Attribute und Geometrietypen haben.

- -d

➔ Löscht zuerst die angegebene Tabelle und verfährt dann wie -c.

- -p

➔ Erstellt nur die notwendige Tabellenstruktur zum Laden des Shapefiles, lädt aber keine Datensätze.

Hinweis: die Parameter -c,-a,-d und -p schließen sich gegenseitig aus.

- -s <SRID>

➔ Legt für die erzeugte Tabelle sowie für alle geladenen Geometrien den SRID fest.

- -r <SRID>

➔ Gibt an, das die Daten des Shapefiles das angegebene Referenzsystem nutzen. Wenn -s ebenfalls angegeben wurde, werden alle Datensätze in dieses System transformiert. Wenn nur -r und -G angegeben sind werden die Datensätze in das Bezugssystem WGS 84 lon/lat (SRID 4326) des Geography Typs konvertiert. Sind weder -s noch -G angegeben verhält sich der Parameter wie -s.

- -I

➔ Erzeugt einen räumlichen Index für die Geometriespalte der erzeugten Tabelle.

Hinweis: Die Generierung der Indizes hat auf dem Testsystem nicht funktioniert und die Erstellung der Tabellen verhindert. Die separate Erstellung von Indizes nach dem Laden (siehe 3.1.6 räumliche Indizes) ist aber ohne Probleme möglich.

- -S

➔ Erzeugt beim Ladevorgang wenn möglich einfache Geometrien anstelle von Multi-Geometrien. Ohne diesen Parameter werden grundsätzlich Multi-Geometrien erzeugt.

- -n

➔ Lädt nur die Attribute aus der *.dbf Datei und ignoriert die Geometrien..

- -G

➔ Lädt die Daten im Geography Typ. Hierfür muss das Shapefile im Bezugssystem WGS 84 lon/lat (SRID 4326) vorliegen.

Für eine vollständige Darstellung aller Parameter kann die eingebaute shp2pgsql Hilfe über den Parameter „-?“ aufgerufen werden.

Die im Tutorial verwendeten Demodaten liegen größtenteils als Shapefiles vor und werden mithilfe von "shp2pgsql" eingelesen.

Anweisungen zum Laden der Tutorial Daten:

```
cd /Pfad_zum_Shapefileordner/Name_des_Shapefileordners

 for shape in $(ls *shp | awk –F "." '{print $1}'); do
/usr/lib/postgresql/8.4/bin/shp2pgsql –c –s 25833 –i $shape public.$shape | psql
–d postgis_tutorial –U pgnutzer; done
```

Diese Anweisung sollte alle Shapefiles im aktuellen Ordner in die Datenbank importieren.

Die for-Schleife dient lediglich dazu, jedes Shapefile im Ordner anzusprechen, sodass es von shp2pgsql geladen werden kann. Das shp2pgsql Programm ist dafür ausgerichtet, nur den Namen der Shapefiles ohne Dateiendung entgegenzunehmen, die zugehörigen Dateien (*.shp,*.dbf,*.shx, und ggf.*.prj) werden automatisch ausgewertet. Die awk Anweisung erfüllt den Zweck, diese Namen ohne Dateiendungen aus der Liste der Dateien im Ordner herzustellen.

Der eigentliche shp2pgsql Aufruf erzeugt eine Reihe von SQL-Anweisungen, die die jeweiligen Shapefiles in die Datenbank abbilden.

Diese SQL-Anweisungen werden über den Unix pipe Operator und psql an die Datenbank postgis_tutorial weiter gegeben, wobei die Identität von pgnutzer für die Anmeldung verwendet wird.

Alternativ kann die Ausgabe von shp2pgsql in eine Datei umgeleitet werden, die dann separat an eine PostgreSQL Client-Applikationen weitergegeben werden kann.

Zum besseren Verständnis der Arbeitsweise des Shapefileimports ist im Folgenden ein Auszug aus den generierten SQL-Anweisungen für das Shapefile „verkehr“ dargestellt:

```
SET CLIENT_ENCODING TO UTF8;
SET STANDARD_CONFORMING_STRINGS TO ON;
BEGIN;
CREATE TABLE "public"."verkehr" (gid serial,
"cat" float8,
"label" varchar(80));
ALTER TABLE "public"."verkehr" ADD PRIMARY KEY (gid);
SELECT AddGeometryColumn('public','verkehr','geom','25833','MULTILINESTRING',2);
INSERT INTO "public"."verkehr" ("cat","label",geom) VALUES
('4','Nebenstrasse','0105000020E964000001000000010200000003000000DFFE1F2FC881174
10A94896931A75641E905FEE1AF8417416235A3CD09A756418C8C9ADCE38417413546C48D08A7564
1');
```

Die Tabellenerstellung unterscheidet sich nur wenig von der oben besprochenen manuellen Variante. Der Großteil der Anweisungen besteht aus einzelnen INSERT Anweisungen für jeden Datensatz des Shapefiles. Wie bereits erwähnt, ist HEXEWKB das räumliche Format, dass direkt in eine Tabelle eingefügt werden kann. Deshalb werden die Geometriedaten von shp2pgsql in diesem Format erzeugt.

Nutzung des Shapefile Exports durch pgsql2shp

Der grundlegende Aufruf von pgsql2shp erfolgt analog zu shp2pgsql nach folgendem Muster:

```
pgsql2shp [Optionen] <Name_der_Datenbank> [<Schemaname>.]<Name_der_Zieltabelle>
```

oder

```
pgsql2shp [Optionen] <Name_der_Datenbank> <Abfrage>
```

In der zweiten Version kann eine Abfrage angegeben werden, deren Ergebnis sofern möglich in ein Shapefile exportiert wird.

Pgsql2shp stellt verschiedene Optionen in Form von Parametern bereit, von denen einige in Folgenden kurz aufgeführt werden sollen:

- -f <Dateiname>

➔ Legt den Dateinamen der exportierten Datei fest.

- -h <Host>

➔ Bestimmt die Hostadresse des Datenbankservers.

- -p <Port>

➔ Legt den Port fest auf dem die Verbindung zur Datenbank läuft.

- -P <Passwort>

➔ Legt das Passwort fest mit dem sich pgsql2shp bei der Datenbank anmeldet.

- -u <Nutzername>

➔ Legt den Nutzernamen fest unter dem sich pgsql2shp bei der Datenbank anmeldet.

Aufruf am Beispiel der zuvor erstellten Tabelle „tutopoly “:

```
/usr/lib/postgresql/8.4/bin/pgsql2shp –f tutopoly –h localhost –p 5432 –u
pgnutzer –P PostGISVM –r postgis_tutorial public.tutopoly
```

3.2.5 Interaktion mit GIS-Anwendungen am Beispiel von gvSIG

GvSIG bietet die Möglichkeit, Datensätze aus PostGIS Datenbanken zu importieren und nach PostGIS zu exportieren. Auch neu erstellte Layer, z.B. durch die Digitalisierung von Vektordaten aus Luftbildern oder aus anderen Dateiformaten, wie ESRI Shapefiles oder *.dxf Dateien, können in die Datenbank exportiert werden.

Im Folgenden werden die notwendigen Schritte dafür aufgezeigt.

Vorbereitung

➔ Erstellen einer neuen Ansicht im Projektmanager:

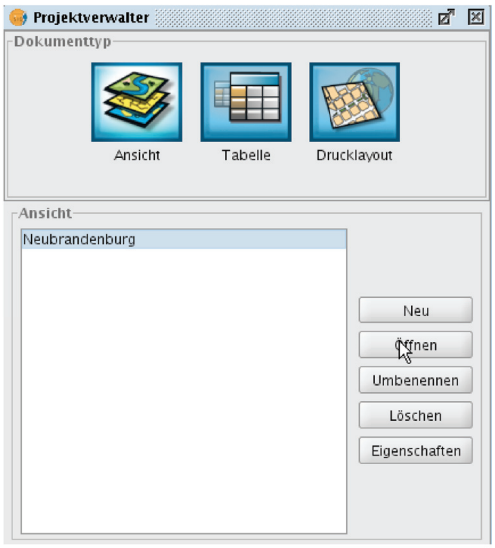

Abbildung 1: gvSIG Projektmanager

➔ Im Ansichtsfenster wird die DB-Verwaltung gestartet, um die PostGIS Datenbank anzumelden:

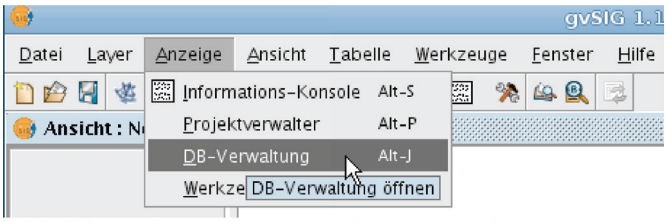

Abbildung 2: gvSIG Datenbankverwaltung

➔ In der Datenbankverwaltung werden eine neue Verbindung erstellt und die notwendigen Daten zum Verbindungsaufbau mit der Datenbank postgis_tutorial eingetragen:

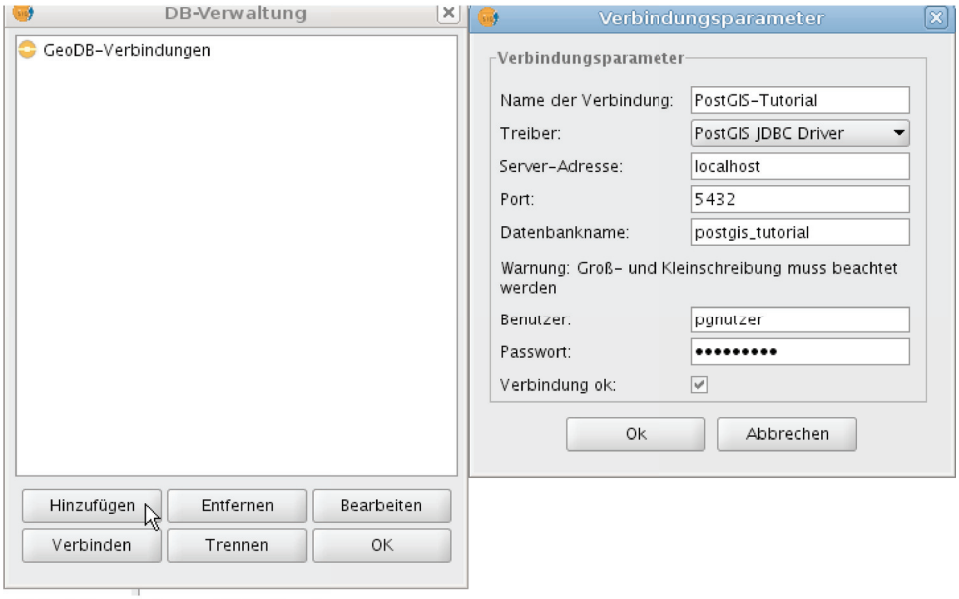

Abbildung 3: gvSIG Definieren einer Datenbankverbindung

➔ Bei erfolgreicher Verbindung erscheint die Verbindung in der DB-Verwaltung. Auf diese Weise können mehrere Datenbanken angebunden werden:

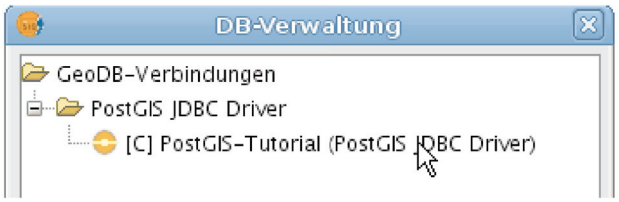

Abbildung 4: gvSIG erfolgreich eingebundene Datenbankverbindung

➔ Die Ansicht verfügt über ein Bezugssystem, das an die zu ladenden Daten angepasst werden sollte. Dies geschieht über die Ansichtseigenschaften:

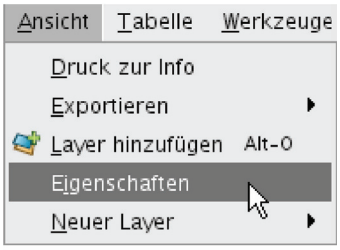

Abbildung 5: gvSIG "Ansicht" Menü

➔ In diesem Feld können verschiedene Einstellungen zu den Karteneinheiten und der Projektion gemacht werden:

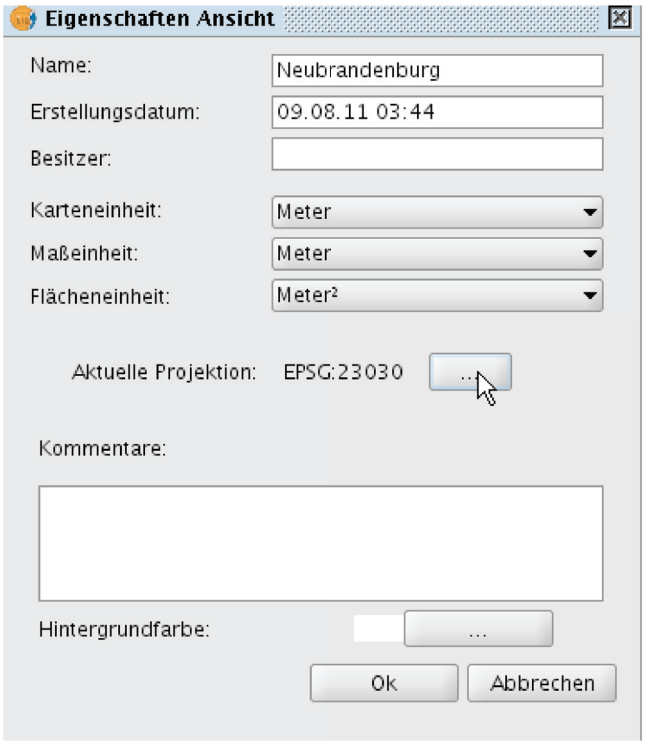

Abbildung 6: gvSIG "Eigenschaften" Menü der Ansicht

➔ Die Projektion wird auf den Wert der Tutorialdaten eingestellt. Da der EPSG Code bekannt ist, kann die gewünschte Projektion schnell gefunden werden:

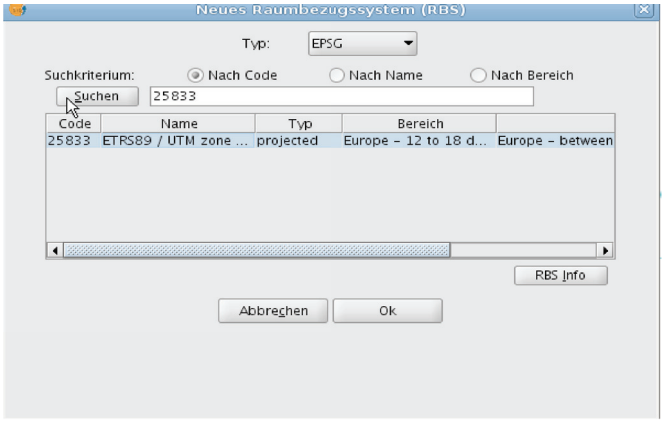

Abbildung 7: gvSIG Auswahlmenü für Raumbezugssysteme

Import von PostGIS Datensätzen

➔ Layer können über das „Menu Layer hinzufügen“ aus verschiedenen Datenquellen geladen werden:

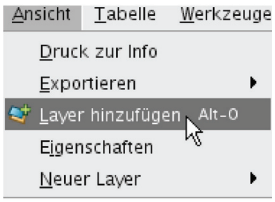

Abbildung 8: gvSIG "Ansicht" Menü

➔ Über den Reiter GeoDB kann die gewünschte Datenbankverbindung ausgewählt werden. Hier sollen alle als Shapefile eingelesenen Datensätze geladen werden:

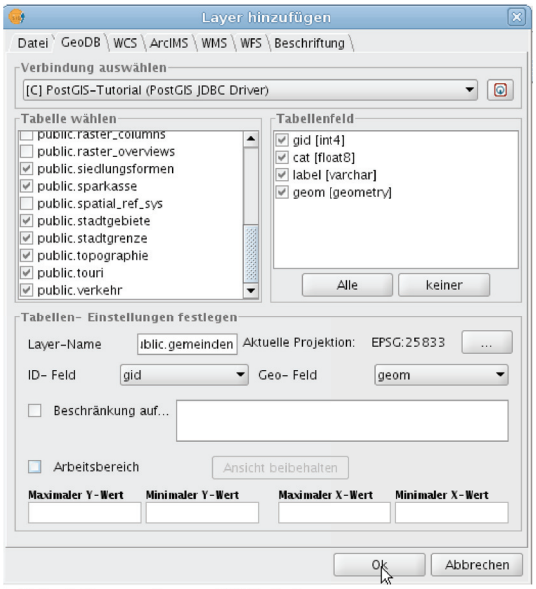

Abbildung 9: gvSIG Menü zum Laden von Datensätzen aus PostGIS

➔ Die aus PostGIS geladenen Datensätze können zusammen mit anderen Datenquellen wie Shapefiles dargestellt und verarbeitet werden:

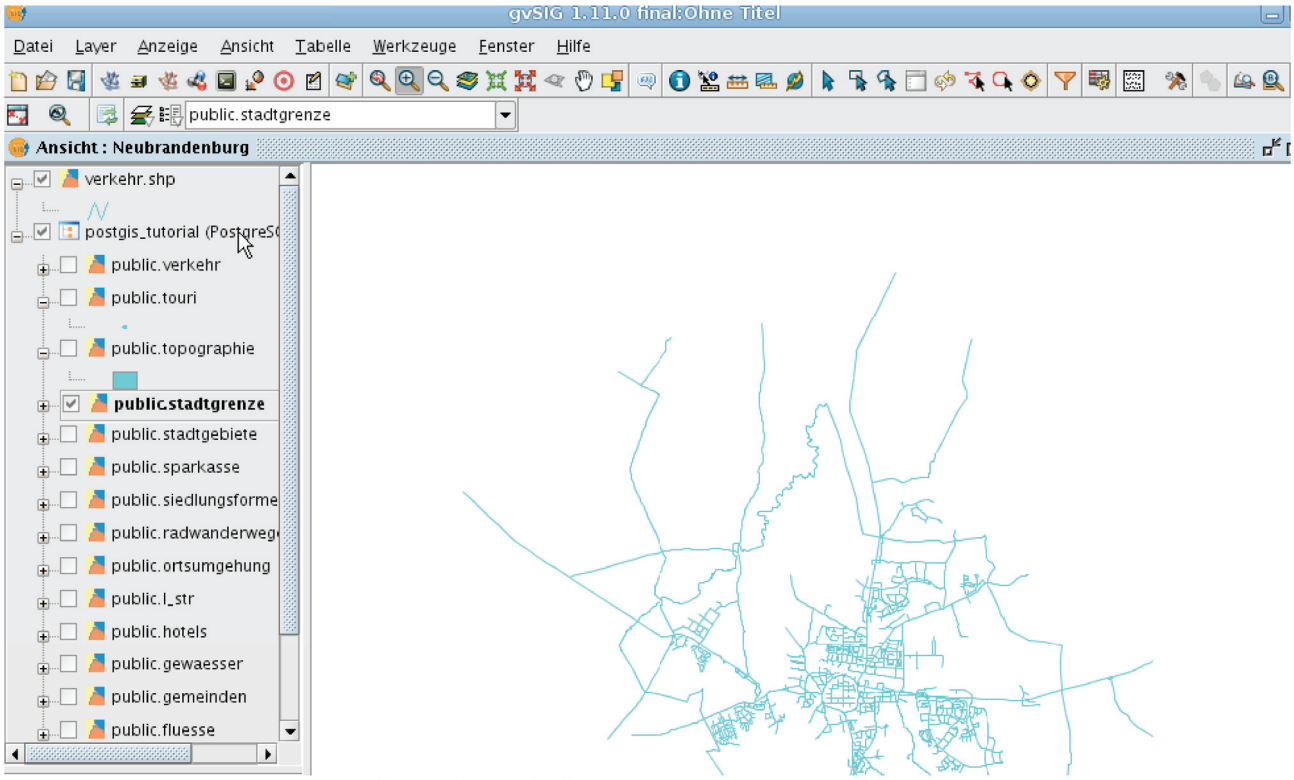

Abbildung 10: gvSIG Darstellung der geladenen Datensätze

Export von gvSIG Daten nach PostGIS

➔ Zuerst wird der zu exportierende Datensatz aus einer anderen von gvSIG lesbaren Datenquelle geladen. Hier wird das Shapefile „verkehr“ geladen:

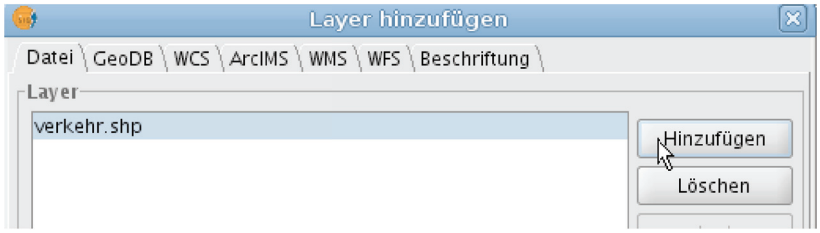

Abbildung 11: gvSIG Hinzufügen eines Shapefile-Layers

➔ Dieser Layer wird zunächst als aktiver ausgewählt und kann dann über das dargestellte Menü in verschiedene Formate, darunter PostGIS, exportiert werden:

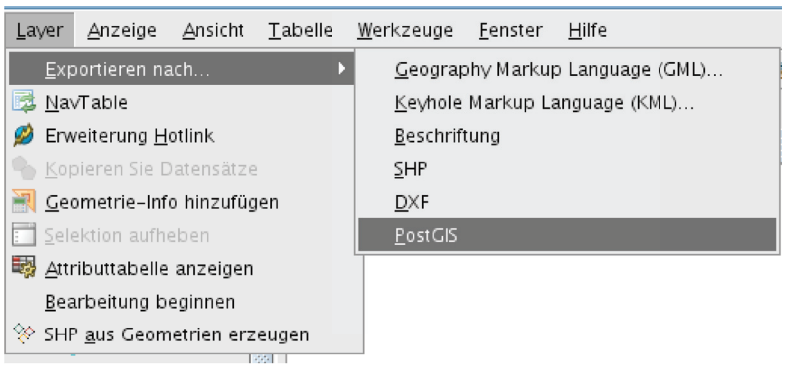

Abbildung 12: gvSIG Menü zum Exportieren eines Layers nach PostGIS

➔ Der neue Layer muss benannt werden. Die Erfahrungen bei der Erstellung der Beispiele haben gezeigt, dass die Verwendung von Großbuchstaben hier zu Schwierigkeiten bei Abfragen gegen die neue Tabelle führen kann. Daher wurden alle Layer in Kleinschreibung benannt:

Abbildung 13: gvSIG Benennung der neuen Tabelle

Nachdem die Verbindung zur gewünschten PostGIS Datenbank hergestellt wurde, ist wird der Export durchgeführt:

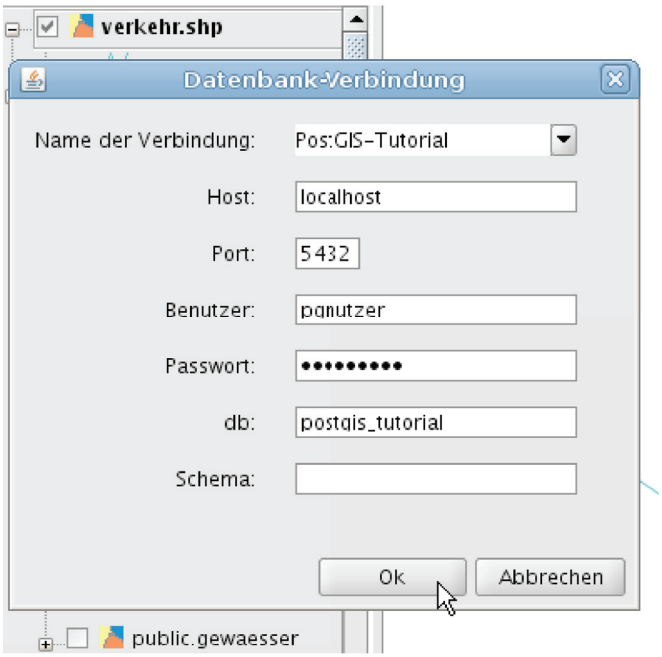

Abbildung 14: gvSIG Menü zur Bestätigung der Datenbankverbindung für den Export

Erzeugen von PostGIS Tabellen in gvSIG

➔ GvSIG erlaubt es, neue PostGIS Layer zu erstellen. Dies kann z.B. verwendet werden, wenn Objekte digitalisiert werden und die Speicherung des Datenbestandes in einer PostGIS Datenbank stattfinden soll:

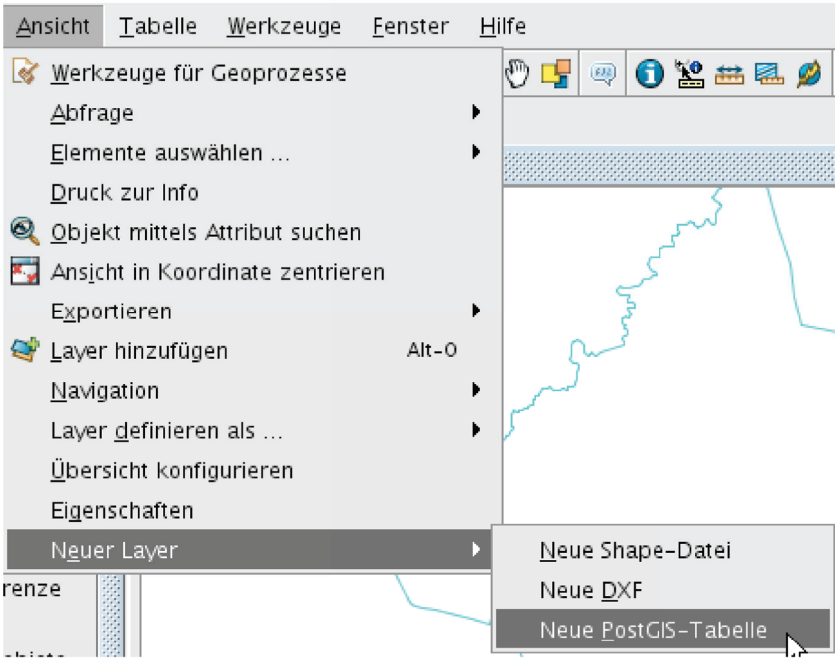

Abbildung 15: gvSIG Menü zur Erstellung einer neuen PostGIS Tabelle

➔ Es werden verschiedene Geometrietypen für den neuen Layer unterstützt, wobei hier die Entscheidung auf Polygondaten fallen soll:

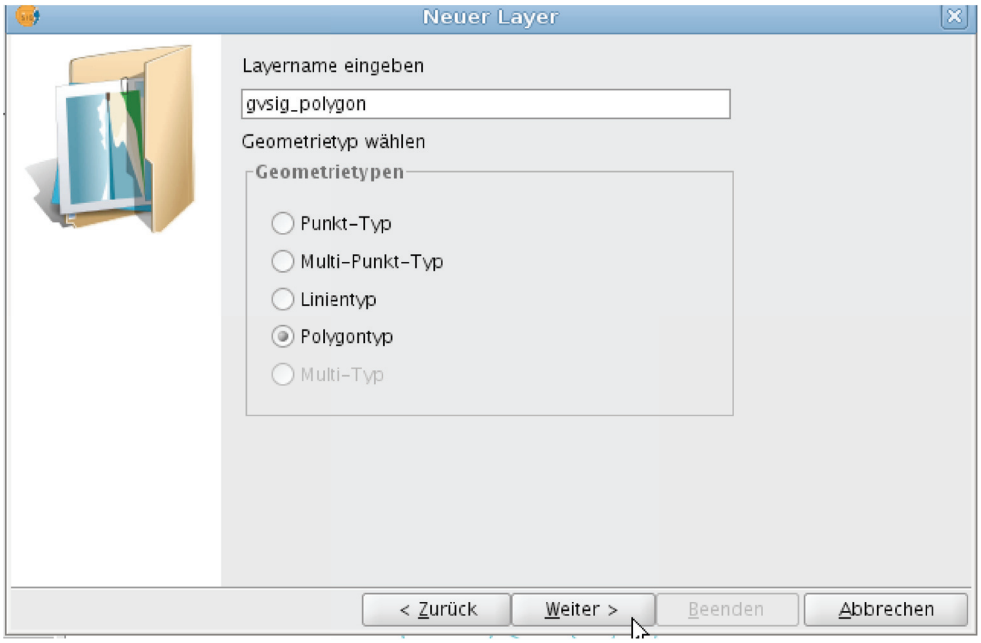

Abbildung 16: gvSIG Menü zur Bestimmung des Geometrietyps

➔ Im nächsten Schritt können Attributfelder für die einzelnen Geoobjekte festgelegt werden:

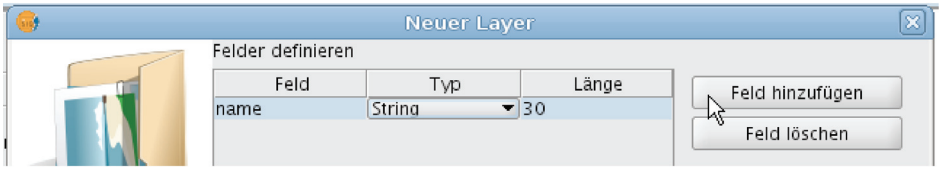

Abbildung 17: gvSIG Menü zur Einrichtung der Datenfeder der neuen Tabelle

➔ Im letzten Schritt muss die Verbindung zur gewünschten Datenbank hergestellt werden:

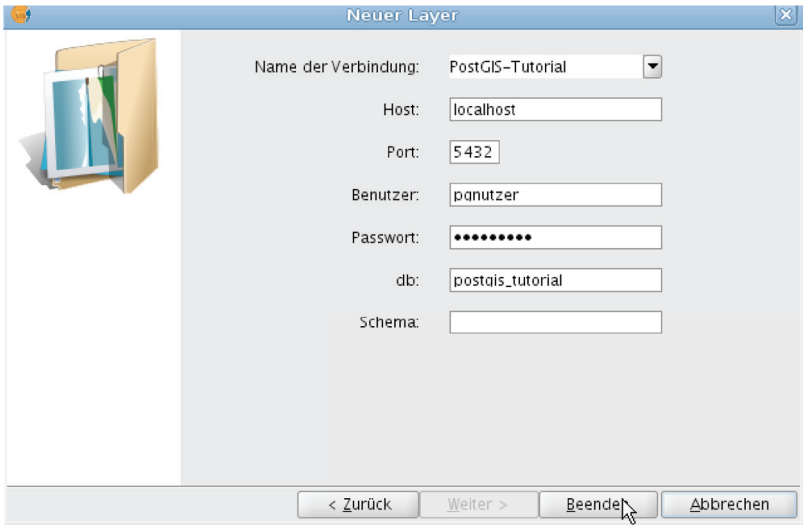

Abbildung 18: gvSIG Menü zu Bestätigung der Datenbankverbindung

➔ Der neue Layer wird im Editierungsmodus geöffnet und es wurden testweise einige Polygone erstellt. Grundsätzlich lassen sich auch importierte PostGIS Layer editieren und mit den Veränderungen in die Datenbank zurückschreiben:

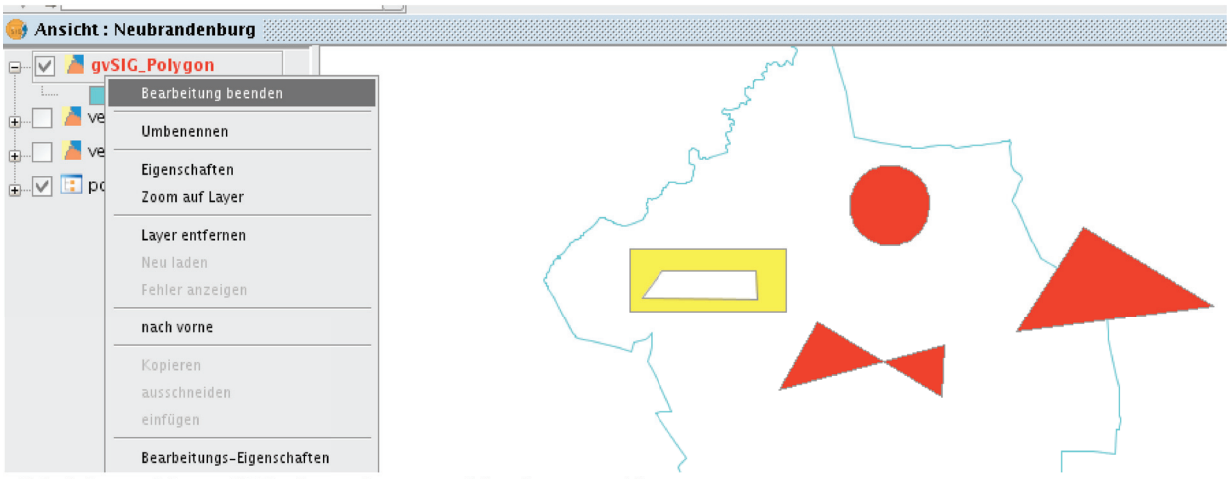

Abbildung 19: gvSIG digitalisierte Objekte zur Abspeicherung in der neuen Tabelle

Sobald die Editierung beendet ist, werden die Daten in die entsprechende PostGIS Tabelle exportiert:

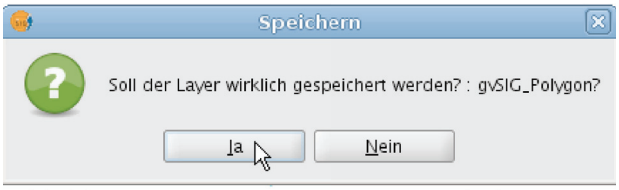

Abbildung 20: gvSIG Bestätigungsdialog zur Speicherung in PostGIS

3.3 Übersicht der PostGIS Funktionen

PostGIS bietet eine große und weiter wachsende Zahl von Funktionen für den Umgang mit geometrischen Daten. Die offizielle Dokumentation des Projekts enthält im Kapitel 7 eine komplette Referenz aller dokumentierten Funktionen, von der aus auch detaillierte Beschreibungen und teilweise Anwendungsbeispiele zugänglich sind.

Die Funktionen sind weitgehend aus diesem Kapitel entnommen worden.

Die hier dargestellten Gruppierungen wurden im Rahmen dieses Tutorials vorgenommen und sollen dem Zweck dienen, Funktionen für verschiedene praktische Aufgabenbereiche leichter auffindbar zu machen. Dabei ist eine scharfe inhaltliche Abgrenzung teilweise nicht möglich, sodass manche Funktionen mehrfach aufgeführt wurden.

3.3.1 Verwaltungsfunktionen

Diese Gruppe enthält Funktionen, die hauptsächlich der Abfrage von Versionsdaten der verschiedenen Komponenten u.ä. dienen.

PostGIS_Full_Version
PostGIS_GEOS_Version
PostGIS_LibXML_Version
PostGIS_Lib_Build_Date
PostGIS_Lib_Version
PostGIS_PROJ_Version
PostGIS_Scripts_Build_Date
PostGIS_Scripts_Installed
PostGIS_Scripts_Released
PostGIS_Uses_Stats
PostGIS_Version

3.3.2 Funktionen zum Umgang mit der GEOMETRY COLUMS Metadatentabelle

Diese Gruppe enthält die weiter oben unter 3.1.3 besprochenen Funktionen für die konsistente Haltung der Metadaten.

AddGeometryColumn
DropGeometryColumn
DropGeometryTable
Populate_Geometry_Columns
Probe_Geometry_Columns
UpdateGeometrySRID

3.3.3 Interface-Funktionen zum Einlesen von Dateiformaten

Diese Gruppe enthält Funktionen zur Erstellung von PostGIS Geometrien aus den unterstützten Eingabeformaten wie WKT oder GML (siehe Abschnitte über das Laden von Daten).

ST_GeogFromText
ST_GeographyFromText
ST_GeogFromWKB
ST_GeomCollFromText
ST_GeomFromEWKB
ST_GeomFromEWKT
ST_GeometryFromText
ST_GeomFromGML
ST_GeomFromKML
ST_GMLToSQL
ST_GeomFromText
ST_GeomFromWKB
ST_WKBToSQL
ST_WKTToSQL

ST_LineFromText
ST_LineFromWKB
ST_LinestringFromWKB
ST_MPointFromText
ST_MPolyFromText
ST_PointFromText
ST_PolygonFromText

3.3.4 Ausgabefunktionen

Diese Gruppe enthält die Funktionen für die Ausgabe von Geometrien in den unterstützten Formaten.

ST_AsBinary
ST_AsEWKB
ST_AsEWKT
ST_AsGeoJSON
ST_AsGML
ST_AsHEXEWKB
ST_AsKML
ST_AsSVG
ST_AsX3D
ST_GeoHash
ST_AsText
ST_AsLatLonText

3.3.5 Referenzsystembezogene Funktionen

Diese Gruppe enthält alle Funktionen mit Bezug zu räumlichen Referenzsystemen und der Interaktion mit der SPATIAL_REF_SYS Tabelle.

ST_SRID
ST_SetSRID
Find_SRID
UpdateGeometrySRID
ST_Transform

3.3.6 Validierungsfunktionen

Diese Gruppe beinhaltet alle Funktionen im Zusammenhang mit der Validierung von Geometrien. (siehe 3.1.5 Validierung)

ST_IsSimple
ST_IsValid
ST_IsValidReason
ST_IsValidDetail
ST_MakeValid
ST_ForceRHR

3.3.7 Informationsabfrage einzelner Objekte

Diese Gruppe beinhaltet Funktionen, die Informationen über die ihnen übergebenen Geometrien zurückgeben. Dabei wurden zwei Untergruppen gebildet.

Die erste fasst Funktionen zusammen, die Informationen über Eigenschaften bzw. Zustände der Geometrien liefern.

Die zweite enthält Funktionen, deren Rückgabewert Geometrien darstellen.

Rückgabe von Zuständen/Eigenschaften

GeometryType
ST_GeometryType
ST_CoordDim
ST_Dimension
ST_HasArc
ST_IsClosed
ST_IsCollection
ST_IsEmpty
ST_IsRing
ST_IsSimple
ST_IsValid
ST_IsValidReason
ST_IsValidDetail
ST_M
ST_Mem_Size
ST_NDims
ST_NPoints
ST_NRings
ST_NumGeometries
ST_NumInteriorRings
ST_NumInteriorRing
ST_NumPatches
ST_NumPoints
ST_SRID
ST_X
ST_Y
ST_Z
ST_Zmflag

Rückgabe von Geometrien

ST_Boundary
ST_Envelope
ST_ExteriorRing
ST_InteriorRingN
ST_GeometryN
ST_PatchN
ST_PointN
ST_StartPoint
ST_EndPoint
ST_Dump
ST_DumpPoints
ST_DumpRings

3.3.8 grundlegende Objekterzeugerfunktionen

Diese Gruppe enthält Funktionen zur Erstellung einfacher Objekte aus Koordinateneingaben oder Geometrien niedrigerer Dimension.

ST_LineFromMultiPoint
ST_MakeLine
ST_MakePolygon
ST_MakePoint
ST_MakePointM
ST_Point
ST_Polygon

3.3.9 Konvertierfunktionen von Objekttypen

Diese Gruppe enthält Funktionen zur Konvertierung von Geometrien in andere Koordinatenformate und Geometrietypen.

ST_Accum
ST_Collect
ST_Force_2D
ST_Force_3D
ST_Force_3DZ
ST_Force_3DM
ST_Force_4D
ST_Force_Collection
ST_ForceRHR
ST_LineMerge
ST_CollectionExtract
ST_CurveToLine
ST_Multi
ST_Reverse
ST_Dump
ST_DumpPoints
ST_DumpRings

3.3.10 Funktionen zur Manipulation von Geometrien

Diese Gruppe stellt verschiedene Funktionen zur Manipulation von Geometrien zur Verfügung. Die Funktionen geben in der Regel die geänderte Geometrie als Rückgabewert aus.

ST_AddPoint
ST_Affine
ST_RemovePoint
ST_Rotate
ST_RotateX
ST_RotateY
ST_RotateZ
ST_Scale
ST_Segmentize
ST_SetPoint
ST_SnapToGrid
ST_Snap
ST_Transform
ST_Translate
ST_TransScale
ST_FlipCoordinates
ST_MakeValid
ST_RemoveRepeatedPoints
ST_Shift_Longitude
ST_Simplify
ST_SimplifyPreserveTopology

3.3.11 Funktionen zur Erstellung abgeleiteter Geometrien

Die Funktionen dieser Gruppe erzeugen neue Geometrien, die unter Eingabe von Parametern anhand der Ursprungsgeometrien erstellt werden.

Die Abgrenzung zu den Verschneidungs- und Geometriemanipulationsfunktionen ist hier fließend. Der Unterschied besteht darin, dass anders als bei der Verschneidung nur eine Geometrie verarbeitet wird und anders als bei der Manipulation tendenziell das Ziel nicht darin besteht, die Ursprungsgeometrie durch die Ausgabegeometrie zu ersetzen.

ST_Buffer
ST_BuildArea
ST_Centroid
ST_ConcaveHull
ST_ConvexHull
ST_MakeEnvelope
ST_MinimumBoundingCircle
ST_Polygonize
ST_Segmentize
ST_Simplify
ST_SimplifyPreserveTopology

3.3.12 Verschneidungsfunktionen zweier Geometrien

Die Funktionen dieser Gruppe dienen dazu, zwei oder mehr Ausgangsgeometrien zu verschneiden, sodass eine neue Geometrie entsprechend der jeweiligen Verschneidungsregeln vorliegt.

ST_Difference
ST_SymDifference
ST_Intersection
ST_MemUnion
ST_Union
ST_UnaryUnion
ST_SharedPaths
ST_Split

3.3.13 Funktionen zur Ermittlung der räumlichen Beziehung zwischen Objekten

In dieser Gruppe wurden zwei Untergruppen gebildet. Die erste enthält Funktionen, die zwei Geometrien auf eine bestimmte räumliche Beziehung testen und booleschen Wert zurückgeben. Die meisten Funktionen dieser Gruppe enthalten eine Implizite Abfrage der Bounding Boxen unter Nutzung vorhandener räumlicher Indizes.

Die zweite Gruppe enthält Funktionen, die charakteristische Geometrien als Ergebniswert zurückgeben.

Rückgabe von booleschen Werten

ST_3DDWithin
ST_3DDFullyWithin
ST_3DIntersects
ST_Contains
ST_ContainsProperly
ST_Covers
ST_CoveredBy
ST_Crosses
ST_LineCrossingDirection
ST_Disjoint
ST_DFullyWithin
ST_DWithin
ST_Point_Inside_Circle
ST_Equals
ST_Intersects
ST_Overlaps
ST_Relate
ST_RelateMatch
ST_Touches
ST_Within

Rückgabe von Geometrien

ST_3DClosestPoint
ST_3DLongestLine
ST_3DShortestLine
ST_ClosestPoint
ST_LongestLine

3.3.14 Messfunktionen

Diese Gruppe enthält Funktionen, die verschiedene Größen im Bezug auf Geometrien messen. Die Funktionen der ersten Untergruppe verarbeiten einzelne Geometrien als Argumente und die der Zweiten erfordern immer zwei Geometrien.

Bezug auf eine Geometrien

ST_Area
ST_Length
ST_Length2D
ST_3DLength
ST_Length_Spheroid
ST_Length2D_Spheroid
ST_3DLength_Spheroid
ST_Perimeter2D
ST_3DPerimeter

Bezug auf zwei Geometrien

ST_3DDistance
ST_3DMaxDistance
ST_Azimuth
ST_Distance
ST_HausdorffDistance
ST_MaxDistance
ST_Distance_Sphere
ST_Distance_Spheroid

3.3.15 Funktionen für Lineares Referencing

Die Funktionen dieser Gruppe beziehen sich auf Techniken des Linearen Referencings, einer Technik bei der Orte durch ihre Position auf einem linienhaften Objekt referenziert werden. Die Funktionen verarbeiten daher hauptsächlich Linienkoordinaten und nutzen außerdem vielfach die „M“ Dimension von 3DM und 4D Koordinaten.

ST_Line_Interpolate_Point
ST_Line_Locate_Point
ST_Line_Substring
ST_Locate_Along_Measure
ST_Locate_Between_Measures
ST_LocateBetweenElevations
ST_AddMeasure

3.3.16 Bounding Box Funktionen

Die Funktionen dieser Gruppe betreffen hauptsächlich die Bounding Boxen, also die kleinsten umschließenden Rechtecke von Geometrien. Die Untergruppe der Räumlichen Operatoren dient der Prüfung von räumlichen Beziehungen zwischen den Bounding Boxen zweier Geometrien. Diese Prüfungen machen Gebrauch von den räumlichen GIST Indizes und können in Abfragen eingesetzt werden, um die Performance zu verbessern. Da seit PostGIS 1.3 die meisten Funktionen zur Prüfung räumlicher Beziehungen implizit Bounding Box Abfragen enthalten, müssen diese Operatoren bei vielen üblichen Abfragearten nicht mehr explizit eingesetzt werden.

ST_MakeBox2D
ST_3DMakeBox
Box2D
Box3D
ST_XMax
ST_XMin
ST_YMax
ST_YMin
ST_ZMax
ST_ZMin
ST_Estimated_Extent
ST_Expand
ST_Extent
ST_3DExtent

Räumliche Operatoren

&&
&&&
&<
&<|
&>
<<
<<|
= -
>>
@
|&>
|>>
~
~=

3.4 Beispielszenarien

3.4.1 Transformation von Datensätzen in ein anderes Bezugssystem

Fallbeispiel: Für die Erstellung eines PostGIS Tutorials werden Beispieldatensätze benötigt und der Großteil der verfügbaren Datensätze liegt in der Form ETRS89/UTM 33N bzw. EPSG 25833 vor. Dieses Bezugssystem soll für alle Datensätze verwendet werden. Einige Datensätze liegen jedoch in einem anderen Bezugssystem (EPSG28403) vor.

```
SELECT ST_AsEWKT(the_geom) AS Original, AsEWKT(ST_Transform((the_geom),25833))
AS Transformiert
FROM b_str_krassowski
limit 3
```

original	transformiert
SRID=28403;MULTILINESTRING((3387990.27359882 5938	SRID=25833;MULTILINESTRING((388036.938467948 5935
SRID=28403;MULTILINESTRING((3391272.04402708 5938	SRID=25833;MULTILINESTRING((391317.341658883 5936
SRID=28403;MULTILINESTRING((3403887.8435196 59494	SRID=25833;MULTILINESTRING((403927.885150347 5946

In PostGIS steht die `ST_Transform` zur Verfügung, mit der Datensätze aller in der SPATIAL_REF_SYS hinterlegten Bezugssysteme ineinander transformiert werden können.

Sollen die transformierten Daten regelmäßig abgerufen werden, kann wie folgt ein VIEW erzeugt werden:

```
CREATE VIEW b_str_trafo AS
      SELECT  gid, strassenna, AsEWKT(ST_Transform((the_geom),25833)) AS geom
      FROM b_str_krassowski
```

Soll die alte Geometrie ganz ersetzt werden, muss temporär eine neue Geometriespalte mit dem neuen SRID angelegt werden, in die dann die transformierten Geometrien gespeichert werden.

Neue Spalte erstellen:

```
select AddGeometryColumn ('b_str_krassowski
','geom2',25833,'MULTILINESTRING',2);
```

Oder alternativ bei neuen PostGIS Versionen mit GEOMETRY_COLUMNS VIEW:

```
ALTER TABLE b_str_krassowski
      ADD COLUMN geom2 GEOMETRY(MULTILINESTRING,25833)
```

Transformierte Daten einfügen:

```
UPDATE b_str_krassowski
      SET geom2=ST_Transform(the_geom,25833)
```

Anschließend kann die alte Spalte gelöscht und die neue umbenannt werden.

```
ALTER TABLE b_str_krassowski
      DROP COLUMN the_geom;

ALTER TABLE b_str_krassowski
      RENAME COLUMN geom2 TO geom;
```

Hinweis:

Bei älteren Versionen mit einer GEOMETRY_COLUMNS Tabelle ist es möglich, das Update ohne Anlegen einer zweiten Spalte durchzuführen, indem zuerst der SRID bezogene CONSTRAINT gelöscht, dann das Update durchgeführt und dann der CONSTRAINT für das neue Bezugssystem neu gesetzt wird.

```
ALTER TABLE b_str_krassowski
      DROP CONSTRAINT IF EXISTS enforce_srid_the_geom;

UPDATE b_str_krassowski
      SET the_geom=ST_Transform(the_geom,25833);

ALTER TABLE b_str_krassowski
      ADD CONSTRAINT enforce_srid_geom CHECK (st_srid(the_geom) = (25833));

SELECT Populate_Geometry_Columns()
```

Dies sollte aufgrund der Gefahr, zwischenzeitlich Daten verschiedener Bezugssysteme zu speichern, nur

durchgeführt werden, wenn garantiert keine neuen Datensätze während des Prozesses eingefügt werden können.

Für neue PostGIS Versionen existiert eine solche Möglichkeit nicht, da der SRID der Geometrien nicht mehr durch CONSTRAINTs abgesichert wird.

3.4.2 Entfernungs- und Umkreisabfragen

Fallbeispiel: Für die Planung eines Urlaubs in Neubrandenburg sollen die drei Hotels ermittelt werden, die am nächsten am Tollensesee gelegen sind.

Abfrage:

```
SELECT hotels.gid,hotels.label AS
hotel,ST_AsText(hotels.geom),ST_Distance(hotels.geom,gewaesser.geom)::Integer AS
entfernung
FROM gewaesser, hotels
WHERE gewaesser.label ILIKE '%Tollensesee%'
ORDER BY entfernung asc limit 3;
```

gid	hotel	st_astext	entfernung
6	Landhotel Broda	POINT(383405.877788756 5935193.79737516)	334
9	Sporthotel	POINT(384569.913596484 5934863.70286852)	565
3	Parkhotel	POINT(384725.246356295 5935066.48035792)	810

Diese Abfrage demonstriert den einfachen Einsatz einer geometrischen Funktion zur Berechnung der Entfernung verschiedener Geoobjekte.

Die Auswahl der drei Datensätze mit der kürzesten Entfernung erfolgt durch die aufsteigende Sortierung der ermittelten Werte und der Beschränkung auf drei Datensätze mit `limit`.

Die Anweisung `::Integer` stellt die PostgreSQL Kurznotation für eine Typenumwandlung dar. Da die Ausgabe der `ST_Distance` Funktion grundsätzlich eine Fließkommazahl ist, wurde so eine Verringerung der Dezimalstellen herbeigeführt, um die Lesbarkeit des Ergebnisses zu verbessern. Alternativ dazu kann PostgreSQLs `round()` Funktion eingesetzt werden.

Die Längenangaben der Entfernungsberechnung beziehen sich grundsätzlich auf das Bezugssystem der verarbeiteten Geometrien. Bei der UTM Abbildung hier handelt es sich also um Angaben in Metern.

Visualisierung:

Die ermittelten Objekte werden mithilfe von gvSIG visualisiert. Dazu wird wie folgt ein VIEW erstellt, der wie eine normale Tabelle in gvSIG geladen werden kann.

```
CREATE VIEW strandhotel1 AS
      SELECT hotels.gid,hotels.label AS
      hotel,hotels.geom,ST_Distance(hotels.geom,gewaesser.geom)::Integer AS
      entfernung
      FROM gewaesser, hotels
      WHERE gewaesser.label ILIKE '%Tollensesee%'
      ORDER BY entfernung asc limit 3;
```

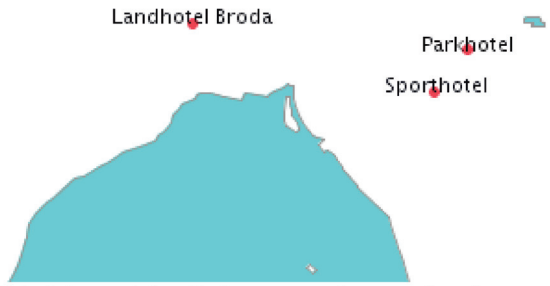

Abbildung 21: Abfrageergebnis - die drei nächstgelegenen Hotels zum Tollensesee

Fallbeispiel:

Alternativ sollen die Hotels ermittelt werden, die nicht weiter als 1500 m vom See gelegen sind.

Abfrage:

```
SELECT hotels.gid,hotels.label AS
hotel,ST_AsText(hotels.geom),ST_Distance(hotels.geom,gewaesser.geom)::Integer AS
entfernung
FROM gewaesser, hotels
WHERE gewaesser.label ILIKE '%Tollensesee%' AND
      ST_DWithin(gewaesser.geom,hotels.geom,1500)
ORDER BY entfernung asc
```

gid	hotel	st_astext	entfernung
6	Landhotel Broda	POINT(383405.877788756 5935193.79737516)	334
9	Sporthotel	POINT(384569.913596484 5934863.70286852)	565
3	Parkhotel	POINT(384725.246356295 5935066.48035792)	810
4	Sankt Georg	POINT(383859.699735857 5935741.05281469)	834
5	Hotel Jahnke	POINT(383999.337191899 5935738.99926834)	843
1	Radisson SAS Hotel	POINT(384762.209429936 5935624.61946701)	1140

Bei diesem Beispiel wird die `ST_DWithin` Funktion verwendet, die allgemein für Abfragen verwendet wird, die feststellen sollen, ob sich ein Objekt in einem bestimmten Umkreis von einem anderen befindet.

Hier erfolgt die räumliche Eingrenzung der Objekte in der WHERE Klausel. Der Aufruf von `ST_Distance` dient hier nur dazu, die einzelnen Entfernungen zur besseren Nachvollziehbarkeit des Beispiels darzustellen.

Visualisierung:

```
CREATE VIEW strandhotel2 AS
      SELECT hotels.gid,hotels.label AS
      hotel,hotels.geom,ST_Distance(hotels.geom,gewaesser.geom)::Integer AS
      entfernung
      FROM gewaesser, hotels
      WHERE gewaesser.label ILIKE '%Tollensesee%' AND
            ST_DWithin(gewaesser.geom,hotels.geom,1500)
      ORDER BY entfernung asc
```

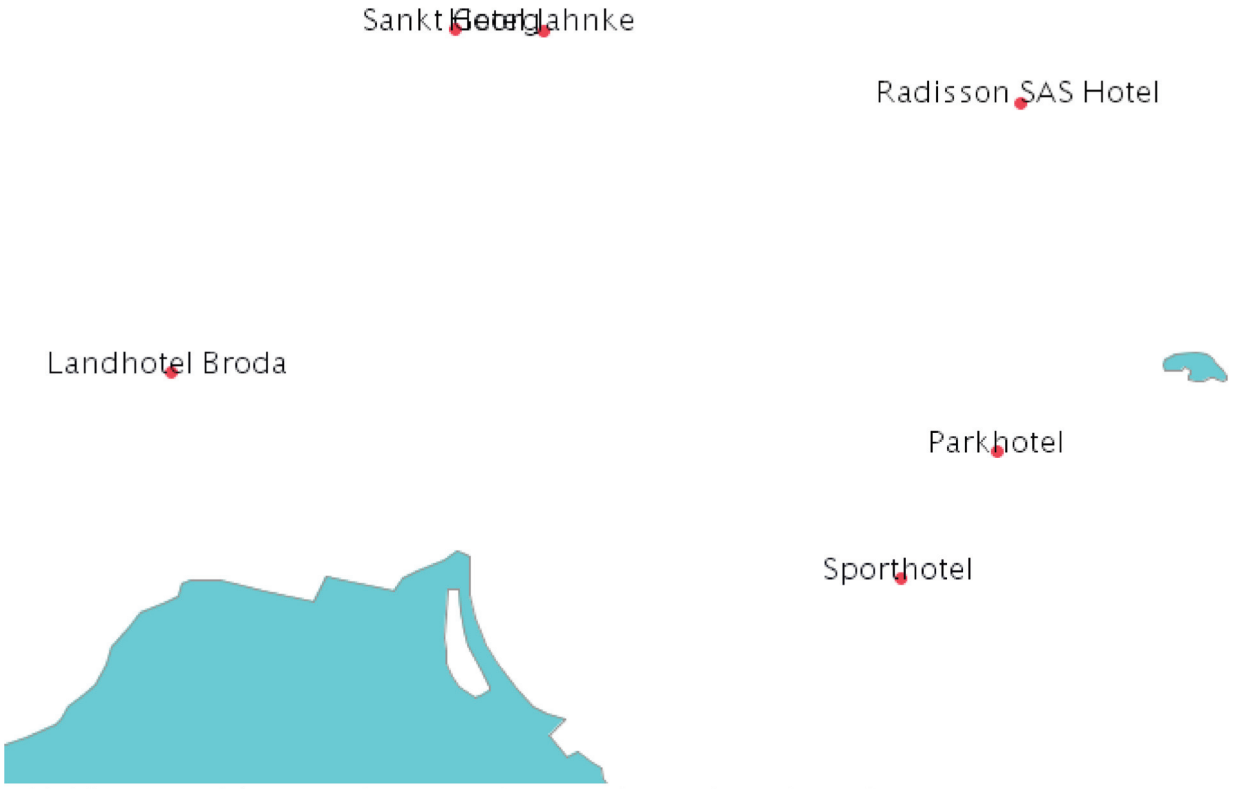

Abbildung 22: Abfrageergebnis - Hotels im 1,5 km Umkreis des Tollensesees

3.4.3 Kombinierte Abfragen von räumlichen Beziehungen und anderen Attributen

Fallbeispiel: Die Gemeinden Neubrandenburg und Burg Stargard wollen im Rahmen der Optimierung der Abfallentsorgung in Erholungsgebieten ermitteln, wie viele Bänke als potentielle Rastplätze mit Müllaufkommen in Waldgebieten stehen und wo diese sich befinden.

```
SELECT count(topographie.geom)
FROM topographie, gemeinden, touri
WHERE touri.label='Baenke' AND (gemeinden.label LIKE '%Neubrandenburg%' OR
gemeinden.label LIKE '%Burg Stargard%') AND topographie.label='Wald'
AND
ST_Intersects(touri.geom,topographie.geom) AND ST_Intersects(touri
.geom,gemeinden.geom)
```

Diese Abfrage verbindet die Geometrien von drei Tabellen, wobei nur die Punkte ausgewählt werden, die in beiden durch `ST_Intersect` abgefragten Polygonen liegen. Gleichzeitig werden gegen alle drei Tabellen attributbasierte Einschränkungen definiert. In dieser Form werden alle Einschränkungen in der WHERE Klausel umgesetzt.

Alternative Ausdrucksmöglichkeit:

```
SELECT count(baenke.geom) FROM
(SELECT label,geom FROM touri WHERE touri.label='Baenke') AS baenke,
(SELECT label,geom FROM topographie WHERE topographie.label='Wald') AS waelder,
(SELECT label,geom FROM gemeinden WHERE gemeinden.label LIKE '%Neubrandenburg%'
OR
gemeinden.label LIKE '%Burg Stargard%') AS NB_BS
WHERE  ST_Intersects(baenke.geom,waelder.geom) AND
ST_Intersects(baenke.geom,NB_BS.geom)
```

Diese Ausdrucksform erzeugt dasselbe Ergebnis wie die obere Abfrage. Die attributbasierten Einschränkungen wurden hier in Unterabfragen in der FROM Klausel umgesetzt. Dabei werden drei virtuelle Tabellen erzeugt, die nur noch die gewünschten Datensätze enthalten.

Ergebnis in beiden Fällen: 38

Visualisierung:

```
CREATE VIEW baenke AS

SELECT baenke.gid,baenke.label,baenke.geom FROM
(SELECT gid,label,geom FROM touri WHERE touri.label='Baenke') AS baenke,
(SELECT label,geom FROM topographie WHERE topographie.label='Wald') AS waelder,
(SELECT label,geom FROM gemeinden WHERE gemeinden.label LIKE '%Neubrandenburg%'
OR gemeinden.label LIKE '%Burg Stargard%') AS NB_BS
WHERE  ST_Intersects(baenke.geom,waelder.geom)
AND
ST_Intersects(baenke.geom,NB_BS.geom)
```

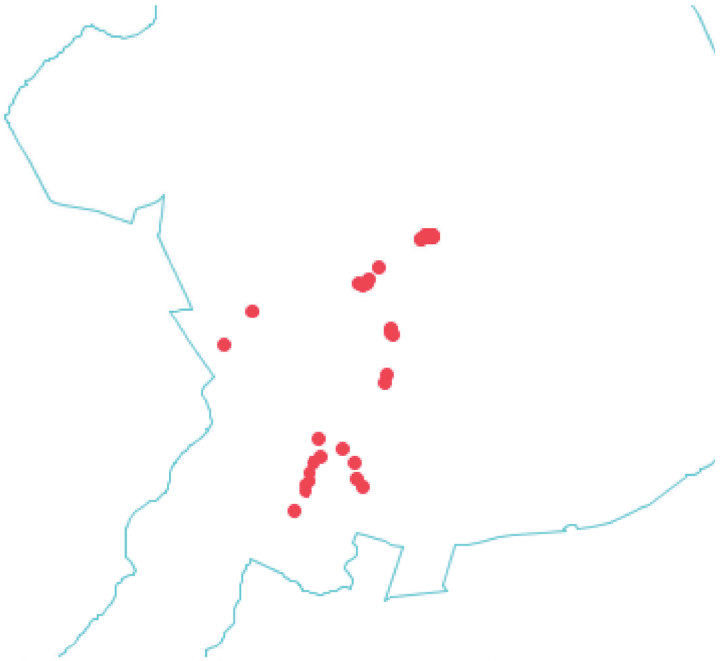

Abbildung 23: Abfrageergebnis - Rastplätze in Waldgebieten

3.4.4 Abfragen unter Nutzung von Abgeleiteten Geometrien und Verschneidungen

Fallbeispiel: Die Stadt Neubrandenburg plant den Bau einer Umgehungsstraße, deren unmittelbare Auswirkungen auf einen Radius von 40 Metern um den Straßenverlauf geschätzt werden. Es soll ermittelt werden, wie viel Fläche je Nutzungsart, entsprechend des Flächennutzungsplans, durch die Auswirkungen der geplanten Straße betroffen sind. Dabei soll auch der Anteil der betroffenen Flächen an den Gesamtflächen der betroffenen Nutzungsarten ermittelt werden.

Buffer Erstellung als Grundlage zur Flächenberechnung und zur Visualisierung

```
CREATE VIEW umgehungbuffer AS

SELECT '40m Buffer'::Text, ST_Buffer(ortsumgehung.geom,40) AS geom FROM
ortsumgehung
```

Die Funktion `ST_Buffer` erzeugt eine Reihe von Polygonen um die Linienkettengeometrie der Ortsumgehung. Diese Polygongeometrie dient der Modellierung des Wirkungsbereiches der Straße und wird weiter unten für die Verschneidung mit den Daten der Flächennutzung eingesetzt. Dabei kann der Buffer sowohl wie hier durch einen VIEW realisiert werden oder mittels entsprechender CREATE TABLE Anweisung als eigene Tabelle erzeugt werden.

Visualisierung:

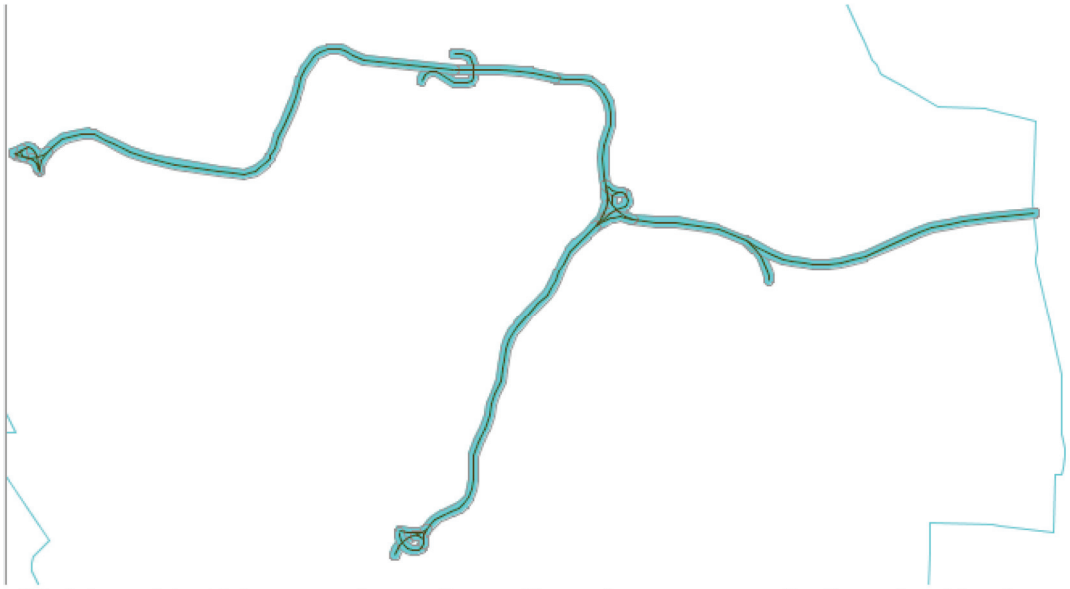

Abbildung 24: Abfrageergebnis - Darstellung des erzeugten Buffers überblendet mit dem ursprünglichen Linienverlauf die Umgehungsstraße

Ermittlung der Flächenanteile

Lösung für einzelne Nutzungsarten:

```
SELECT SUM(ST_AREA(ST_intersection(umgehungbuffer.geom,flaechennutzung.geom )))
FROM umgehungbuffer,flaechennutzung
WHERE label='Bebauung'
```

Diese Abfrage demonstriert das Grundprinzip der Ermittlung der betroffenen Flächen. `ST_intersection` bildet eine neue Geometrie, die den Bereich abdeckt, auf dem sich der Wirkungsbereich der Straße und die Geometrien der Flächennutzungen überschneiden. Die Funktion `St_Area` ermittelt die Fläche der ihr übergebenen Geometrien. Diese Zahlenwerte können dann mithilfe einer normalen SQL Summenfunktion addiert werden.

Schnellere Lösung mit ST_Intersects durch Vorauswahl und GIST Nutzung:

```
SELECT SUM(ST_AREA(ST_intersection(umgehungbuffer.geom,flaechennutzung.geom )))
FROM umgehungbuffer,flaechennutzung
WHERE label='Bebauung' AND
ST_intersects(umgehungbuffer.geom,flaechennutzung.geom)
```

Diese Abfrage wurde um eine Prüfung auf Überschneidung der Geometrien durch `ST_intersects` erweitert. Das Ergebnis ändert sich dadurch nicht, aber die Ausführungsgeschwindigkeit erhöht sich trotz des relativ kleinen Datensatzes merklich.

Die Prüfung durch `ST_intersects` ermöglicht es dem System, eine schnellere Vorauswahl der in Frage kommenden Einzelgeometrien durchzuführen, bei der auch evtl. vorhandene räumliche Indizes genutzt werden können.

Der Aufruf von `ST_Intersection` kann aber nicht eingespart werden, da `ST_Intersects` alle Teilgeometrien auswählt, die sich an irgendeiner Stelle mit der zweiten Geometrie überlappen. Dadurch würden auch die Teile der betreffenden Geometrie mit ausgewählt, die über den Rand der zweiten Geometrie hinausragen, was die Flächenberechnung unbrauchbar machen würde. Beim vorangegangenen Beispiel der Ermittlung aller Bänke in Waldgebieten war ein solches Vorgehen möglich, da die betroffenen Geometrien Punkte waren und somit keine Ausdehnung besitzen.

In dieser Form ist die Abfrage auf einen Geometrie Typ beschränkt. Selbst wenn man mehrere dieser Abfragen in einer größeren kombiniert, müsste man trotzdem alle Flächennutzungen kennen. Dieses Problem wird in der folgenden Version behoben.

Gesamtlösung für die Verschneidung:

```
SELECT clipped.label, SUM(ST_AREA(clipped_geom))
FROM
      (SELECT flaechennutzung.label,
      ST_Intersection(flaechennutzung.geom,umgehungbuffer.geom) As clipped_geom
      FROM umgehungbuffer
      INNER JOIN flaechennutzung
      ON ST_Intersects(flaechennutzung.geom, umgehungbuffer.geom))  As clipped
GROUP BY clipped.label;
```

Diese Lösung stellt eine Abwandlung des Nutzungsbeispiels von `ST_Intersection` in der PostGIS Nutzerdokumentation dar. Es ermöglicht eine korrekte Ermittlung der Flächen und gleichzeitige Zuordnung der Namensfelder für alle vorhandenen Nutzungsarten.

Der größte Teil aller Aktionen findet hier bei der Erzeugung de virtuellen Tabelle in der FROM Klausel statt. Dort wird ein INNER JOIN der beiden Tabellen (bzw. VIEWs) mit den Ausgangsdaten hergestellt. Das Auswahlkriterium dieses JOINs ist die Überschneidungsprüfung durch `ST_Intersects`. Diese Tabelle enthält also nur die Datensätze der Relation Flächennutzung, die tatsächlich von der Verschneidung betroffen sind. In der SELECT Anweisung der Unterabfrage werden die relevanten Werte, also der Name der entsprechenden Nutzungsart sowie das Ergebnis von `ST_Intersection` für die Geometrien ausgewählt.

Dieser Lösung fehlt nun nur noch die Ermittlung und der Vergleich mit den Gesamtflächen.

Gesamtlösung mit Ermittlung und Prozentberechnung der Gesamtflächen:

```
SELECT gesamt.label AS Nutzungsart,ROUND(gesamt.Gesamtflaeche) AS
Gesamtflache_in_qm,ROUND(betroffen.betroffene_Flaechen) AS
betroffene_Flaechen_in_qm,
ROUND((betroffene_Flaechen/Gesamtflaeche*100)::NUMERIC,2) AS Anteil_in_Prozent

FROM
      (SELECT label, SUM(ST_AREA(geom)) AS Gesamtflaeche
      FROM flaechennutzung
      GROUP BY label)

AS gesamt

INNER JOIN

      (SELECT  clipped.label, SUM(ST_AREA(clipped_geom)) AS betroffene_Flaechen
      FROM
            (SELECT flaechennutzung.label,
            ST_Intersection(flaechennutzung.geom,umgehungbuffer.geom) As
            clipped_geom
            FROM umgehungbuffer
            INNER JOIN flaechennutzung
            ON ST_Intersects(flaechennutzung.geom, umgehungbuffer.geom))  As
            clipped
      GROUP BY clipped.label)

AS betroffen

ON gesamt.label=betroffen .label
ORDER BY Gesamt.label
```

Diese Abfrage erzeugt eine zweite virtuelle Tabelle „gesamt“, die alle gewünschten Daten der Gesamtflächennutzung enthält. Diese Tabelle wird in einem INNER JOIN mit der Lösung für die Verschneidung („betroffen“) verbunden. Bedingung ist die Übereinstimmung der Nutzungsarten, da nicht betroffene Nutzungsarten nicht auftauchen sollen.

Die SELECT Anweisung der Abfrage dient der Auswahl und Formatierung der gewünschten Ergebnisspalten sowie der Berechnung des prozentualen Anteils der betroffenen Flächen.

Ergebnis:

nutzungsart	gesamtflache_in_qm	betroffene_flaechen_in_qm	anteil_in_prozent
Bebauung	17927178	467544	2.61
Kleingaerten	4175365	178214	4.27
Wald	11251013	38407	0.34
Wiese	22230667	766641	3.45

Visualisierung der betroffenen Gebiete:

CREATE VIEW betroffene_gebiete AS

```
SELECT clipped.gid,clipped.label, clipped_geom
FROM
(SELECT flaechennutzung.gid, flaechennutzung.label,
(ST_Dump(ST_Intersection(flaechennutzung.geom,umgehungbuffer.geom))).geom As
clipped_geom
FROM umgehungbuffer
INNER JOIN flaechennutzung
ON ST_Intersects(flaechennutzung.geom, umgehungbuffer.geom))  As clipped;
```

Die Abfrage unterscheidet sich nur dadurch von der Lösung der betroffenen Flächen, das ein „gid“ Feld ausgewählt wurde und die von `ST_Intersection` erzeugte Geometrie durch die Funktion `ST_Dump` in einzelne Polygone zerlegt wird. Diese Änderungen haben lediglich den Zweck die Daten für GIS Anwendungen kompatibler zu gestalten.

Abbildung 25: Abfrageergebnis - Darstellung der betroffenen Flächen der Umgehungsstraße mit farblicher Visualisierung nach Nutzungsarten

4 Rasterunterstützung

Allgemein bestehen Rasterdaten aus einer Reihe von Zellen in einer Matrix, denen jeweils ein oder mehrere Werte zugeordnet sind.

Im GIS Umfeld sind Rasterdaten häufig in Form von Luftbildern, Fernerkundungsaufnahmen oder gescannten Papierkarten zu finden. Diese Art der Daten kann oft relativ kostengünstig für große Gebiete erzeugt werden und steht daher in vielen Gebieten in größerem Umfang zur Verfügung als Vektordaten.

Rasterdaten dienen häufig als Grundlage für Analysen direkt auf Basis der Rasterdaten, für die Herstellung von Vektordaten und zur visuellen Aufbereitung von Karten für Nutzer.

Die Unterstützung für die Speicherung und Analyse von Rasterdaten ist in Postgis-2.0 erstmals in der Kerninstallation enthalten. In früheren Versionen musste die optionale Erweiterung WKTRaster installiert werden. Dieses Projekt ist unter dem Namen PostGIS Raster vollständig in die Rasterfunktionalität von Postgis-2.0 integriert worden und wird in Zukunft nicht mehr separat weiterentwickelt.[vgl. 04]

Das Projekt WKTRaster hat zum Ziel, die Rasterunterstützung durch PostGIS so kompatibel wie möglich mit dem Geometry Typ, also der Vektordatenunterstützung zu halten. In diesem Rahmen sollen viele der Funktionen zur Vektoranalyse genauso auf Raster oder gemischte Datensätze anwendbar sein. [vgl. 04]

PostGIS Raster benötigt für einige Funktionen die GDAL Bibliothek und außerdem die GDAL tools für den Export von Bilddateien. GDAL ist ein Open Source Projekt, das insbesondere Formatkonvertierung, aber auch andere Operationen für Rasterdaten ermöglicht.

4.1 Datenstrukturen für PostGIS Raster

4.1.1 Grundlegende Konzepte

- Rasterdaten werden analog zu Geometriedaten in einem einzigen Datentyp „Raster“ ausgedrückt.
- Es ist in PostGIS Rastertabellen möglich, Rasterbilder in Kacheln zu zerteilen. In diesem Fall ist jede Kachel ein eigenständiges Objekt des Datentyps Raster und die Tabelle definiert eine sogenannte „Raster Coverage“, hier also ein Gebiet, das mit Rasterdaten abgedeckt ist.
- Jedes Rasterobjekt kann ein oder mehrere Bänder beinhalten, die ihrerseits wieder Pixelwerte beinhalten.
- Pixelwerte können einem von elf verschiedenen Typen angehören. Dies sind:
  - ➔ 1BB - „1-bit boolean“ Dieser Typ entspricht einer reinen schwarz-weiß Darstellung bzw. einer Unterscheidung von zwei Zuständen bei nicht-bildlichen Rastern.
  - ➔ 2/4/8/16/32 BUI – „X bit unsigned Integer“. Diese Typen entsprechen verschiedenen Farbtiefen bzw. Werteauflösungen, die durch nicht-negative Ganzzahlen ausgedrückt werden.
  - ➔ 8/16/32 BSI – „X bit signed Integer“. Diese Typen entsprechen verschiedenen Farbtiefen bzw. Werteauflösungen, deren Werte positiv oder negativ sein können.
  - ➔ 32/64 BF – „X bit float“. Diese Typen entsprechen verschiedenen Farbtiefen bzw. Werteauflösungen, deren Werte durch Fließkommazahlen ausgedrückt werden.
- Jedes Band einer Raster Spalte hat einen „noData“ Wert. Dies ist ein Pixelwert bzw. eine Spanne von Werten, die von Funktionen gesondert betrachtet werden. Der Wert bestimmt dabei Pixel, die nicht als relevante Informationen eingestuft werden sollen. Ist dieser Wert leer (NULL), wird kein Pixelwert als „noData“ eingestuft.
- Jedes Raster hat eine Ausdehnung (extend), die als ein Polygon definiert ist, welches alle Kacheln des Rasters umfasst.
- PostGIS kann sowohl regulär als auch irregulär gekachelte Raster speichern und verarbeiten. Manche Funktionen, wie das Erstellen von Overviews, stehen jedoch nur für regulär gekachelte (regular blocked) Raster zur Verfügung. Regulär gekachelte Raster müssen verschiedene Bedingungen erfüllen: [vgl. 04]
  - ➔ Alle Kacheln müssen dieselbe Größe aufweisen.
  - ➔ Alle Kacheln müssen auf einem gleichmäßigen Raster angeordnet sein und dürfen sich nicht überlappen.
  - ➔ Die linke obere Kachel des Rasters muss an der linken oberen Ecke der Ausdehnung des Rasters beginnen. Kacheln dürfen am rechten und unteren Rand über die Ausdehnung hinausragen, wobei die herausragenden Teile nicht als Teil des Rasters berücksichtigt werden.
  - ➔ Die Ausdehnung muss durch ein einfaches nicht gedrehtes Rechteck definiert sein.
- Raster können durch PostGIS Raster grundsätzlich innerhalb und außerhalb der Datenbank gespeichert werden. Bei Speicherung innerhalb der Datenbank wird das Raster ganz in einem speziellen WKB Format in der Datenbank gespeichert. Bei Speicherung außerhalb der Datenbank werden lediglich die Metadaten des Rasters in der Datenbank gehalten, während die tatsächlichen Pixeldaten außerhalb in Rasterdateien gespeichert bleiben. Derzeit funktionieren die meisten Analysefunktionen nur für innerhalb der Datenbank gespeicherte Raster.[vgl. 06 S. 377-378]
- Metadaten der Raster wie die Georeferenzierung, das räumlichen Bezugssystem, der NoData Wert oder die Bounding Box sind in den Rasterkacheln selbst, als Teil des Rastertyps gespeichert. Jede

Kachel verfügt also über eigene rasterinterne Metadaten. Die Metadatentabellen beziehen sich auf Metadaten der gesamten Rasterspalte.

- Die Daten zur Georeferenzierung bestehen aus den Werten:
  - ➔ scalex,s und scaley: die reale räumliche Ausdehnung entlang der x- bzw. y- Achse, die durch ein Pixel ausgedrückt wird.
  - ➔ skewx und skewy: Parameter, die eine Drehung bzw, Schiefstellung des Bildes definieren.
  - ➔ upperleftx und upperlefty: die Koordinaten der linken oberen Ecke des Rasters.
  - ➔ Diese Werte sind in sogenannten World Files enthalten, die Georeferenzierungsdaten für Bilddateien speichern.

4.1.2 Die Metadatentabelle RASTER_COLUMNS

Die RASTER_COLUMNS Tabelle ist analog zur GEOMETRY_COLUMNS gestaltet und beinhaltet die Metadaten für alle Spalten mit Daten des Rasterdatentyps.

Die RASTER_COLUMNS beinhaltet folgende Attribute:

- r_table_catalog – Der Name des „catalog“ im Pfad zur Spalte. Dieses Feld enthält in PostGIS immer den Namen der Datenbank, da in PostgreSQL kein Gegenstück zu „catalogs“ existiert.
- r_table_schema – Der Name des Schemas im Pfad zur Spalte.
- r_table_name – Der Name der Tabelle im Pfad zur Spalte.
- r_column – Der Name der Rasterspalte
- srid – Der SRID zur Bestimmung des räumlichen Referenzsystems der Rasterdecke.
- pixel – Ein Array der Pixeltypen mit je einem Element pro Band.
- out_db – Ein boolescher Wert der angibt, ob das Raster innerhalb oder außerhalb der Datenbank gespeichert wird.
- regular_blocking – Ein boolescher Wert der angibt, ob das Raster regulär gekachelt ist.
- nodata_values – Ein Array der NoData Werte für jedes Band des Rasters.
- scale_x – Die Breite in der Maßeinheit des Bezugssystems, die durch ein Pixel ausgedrückt wird.
- scale_y – Die Länge in der Maßeinheit des Bezugssystems, die durch ein Pixel ausgedrückt wird.
- blocksize_x – Die Breite einer Kachel in Pixeln. Der Wert ist bei irregulär gekachelten Rastern leer (NULL).
- blocksize_y – Die Länge einer Kachel in Pixeln. Der Wert ist bei irregulär gekachelten Rastern leer (NULL).
- extent geometry – Die Ausdehnung des Gesamtrasters. Bei regulär gekachelten Rastern handelt es sich immer um ein Rechteck, ansonsten um ein entsprechendes Polygon.

Die RASTER_COLUMNS Tabelle wird ähnlich der alten GEOMETRY_COLUMNS Tabelle nicht automatisch mit dem tatsächlichen Datenbestand abgeglichen. Derzeit existieren drei Funktionen zum Umgang mit den Metadaten.

Die Funktionen sind wie folgt zu beschreiben:

- AddRasterColumn – fügt der Tabelle eine Spalte des Typs Raster hinzu und registriert diese in der Metadatentabelle. Außerdem wird ein CONSTRAINT zu Einhaltung des SRID erstellt.
- DropRasterColumn – löscht die betreffende Rasterspalte und deregistriert diese in der Metadatentabelle

- DropRasterTable – löscht eine Tabelle mit Rasterspalten und deregistriert diese in der Metadatentabelle

4.1.3 Die Metadatentabelle RASTER_OVERVIEWS

Der Begriff Overviews wird von GDAL und auch PostGIS für Kopien eingelesener Raster mit reduzierter Auflösung verwendet. In anderen Systemen spricht man in diesem Zusammenhang von sogenannten Bildpyramiden.

Overviews in PostGIS ermöglichen es, beliebige Verkleinerungsfaktoren für die generierten Bilder zu verwenden. Dabei wird bei jedem Aufruf ein Raster mit der gewünschten Verkleinerungsstufe erzeugt, sodass es nicht notwendig ist, beispielsweise ein halb so großes Bild wie das Ausgangsbild zu erstellen, bevor eine Viertelung der Auflösung vorgenommen werden kann.

Bildpyramiden bzw. Overviews können für verschiedene Zwecke eingesetzt werden. Übliche Einsatzbereiche sind die Bereitstellung von schneller übertragbaren Bildern mit niedrigerer Auflösung während des Ladens der Hauptdatei oder für heraus gezoomte Darstellungen. Räumliche Abfragen können auf die Overviews angewendet werden, um in kurzer Zeit näherungsweise Ergebnisse zu erhalten.[vgl. 09]

Es gibt noch weitere spezielle Anwendungsmöglichkeiten wie z.B. in den Bereichen der Photogrammetrie oder Computer Vision.

Die Overviews sind grundsätzlich eigenständige Tabellen, die aber nicht in der RASTER_COLUMNS Tabelle registriert werden. Stattdessen existiert mit der RASTER_OVERVIEWS eine weitere Metadatentabelle, die die Beziehung zum Originalraster definiert.

Die RASTER_OVERVIEWS beinhaltet folgende Attribute:

- o_table_catalog, o_table_schema, o_table_name, o_column

➔ Diese Attribute definieren zusammen den vollständigen Namen der und Pfad zur Rasterspalte des Overviews.

- r_table_catalog, r_table_schema, r_table_name, r_column

➔ Diese Attribute definieren zusammen den vollständigen Namen der und Pfad zur Rasterspalte des Ausgangsrasters.

- out_db – Ein boolescher Wert der angibt, ob das Raster innerhalb oder außerhalb der Datenbank gespeichert wird.
- overview_factor – Der Verkleinerungsfaktor des Overviews. Ein Faktor von 2 bedeutet dabei, dass der Overview die halbe Größe des Originals hat.

4.2 Speichern und Laden

4.2.1 raster2pgsql.py Rasterimport Skript

Grundlegende Funktion

Aktuell ist das Importskript „raster2pgsql.py“ die hauptsächliche Möglichkeit, Rasterdaten nach PostGIS zu importieren. Neben GDAL, das für PostGIS Raster generell notwendig ist, müssen deshalb auch Python und die Python bindings für GDAL installiert sein.

Langfristig ist geplant, dieses Importskript durch ein kompiliertes, ausführbares Programm analog zu shp2pgsql zu ersetzen. Dadurch könnte die derzeitige Abhängigkeit der PostGIS Raster Nutzung von Python und den entsprechenden GDAL bindings beseitigt werden. [vgl. 10]

Der grundlegende Aufruf von raster2pgsql.py erfolgt nach folgendem Muster:

```
raster2pgsql.py  [Optionen] –r Rastername –t Zieltabellenname
```

Die Zieltabelle wird in der Notation „Schemaname.Tabellenname“ angegeben. Wird kein Schema angegeben, nimmt das Programm an, dass die Tabelle im Schema „public“ liegt.

Laut der Dokumentation des Skriptes ist es möglich, für „Rastername“ Platzhalter wie in einer UNIX Shell üblich (z.B. ? und *) zu verwenden, um z.B. mehrere Dateien auf einmal zu laden. Dies hat im Testsystem nicht funktioniert. Ob es sich dabei um einen Fehler im Importskript oder ein Problem mit dem Testsystem selbst handelt, konnte nicht ermittelt werden.

raster2pgsql.py stellt verschiedene Optionen in Form von Parametern bereit, von denen einige in Folgenden kurz aufgeführt werden sollen:

- -c

➔ Erzeugt eine neue Tabelle mit Rasterspalte und füllt sie mit den Daten des Rasters. Dies ist die Standardeinstellung, wenn keiner der drei Parameter -c,-a oder -d gegeben ist.

- -a

➔ Fügt die geladenen Daten zu denen einer vorhandenen Tabelle hinzu.

- -d

➔ Löscht zuerst die angegebene Tabelle und verfährt dann wie -c.

Hinweis: die Parameter -c,-a, und -d schließen sich gegenseitig aus.

- -s <SRID>

➔ Legt das Bezugssystem (SRID) für die erzeugte Rasterspalte fest.

- -b <BAND>

➔ Bestimmt das Band des Rasters, das geladen werden soll. Ohne diesen Parameter werden alle Bänder des Rasters geladen.

- -k BLOCK_SIZE

➔ Zerlegt das Raster in Kacheln. Die Größe der Kacheln wird im Format BREITExHÖHE angegeben, wobei die beiden Maße in Pixeln angegeben werden z. B. 200x200.

- -R

➔ Bewirkt, dass das Raster außerhalb der Datenbank gespeichert wird. In der Datenbank werden dann nur die Metadaten und der Pfad zur Rasterdatei gespeichert.

- -l OVERVIEW_LEVEL

➔ Bewirkt, dass ein Overview des Rasters gespeichert wird. Die Tabelle, die das Overview enthält, wird nach dem Muster o_<LEVEL>_<TABELLENNAME> benannt. Overviews stehen nur für regulär gekachelte Raster zur Verfügung.

Hinweis: Zum Erstellungszeitpunkt des Tutorials ist die Overview Erstellung durch raster2pgsql.py defekt, sodass sie nur durch manuelles Editieren des generierten SQL Skripts möglich ist.[vgl. 11]

- -F

➔ Fügt der erzeugten Rastertabelle ein Attribut „filename“ hinzu, das den Dateinamen des Quellrasters enthält. Dies kann z. B. Bei Rastertabellen, die aus mehreren Dateien zusammengefügt worden von Nutzen sein.

- -I

➔ Erzeugt einen räumlichen Index für die Rasterspalte. Hinweis: Wurden einer Spalte später Daten mit der -a Option hinzugefügt werden, so musste der Index vorher entfernt werden. Alternativ kann der Index jederzeit über normale SQL Anweisungen für bestehende Tabellen erzeugt werden.

- -o <FILENAME>

➔ Erzeugt eine Datei mit dem übergebenen Namen, in der die generierten Anweisungen enthalten sind.

Für eine vollständige Darstellung aller Parameter kann die eingebaute raster2pgsql.py Hilfe über den Parameter „-h“ aufgerufen werden.

Laden der Tutorialdaten

Laden des Landsat Bildes neubrandenburg.tif:

```
cd /Pfad_zu_den_Bilddateien/

python /usr/lib/postgresql/8.4/bin/raster2pgsql.py –s 25833 –k 200x200  –r
neubrandenburg.tif –F –t public.landsat –I | psql –d postgis_tutorial –U
pgnutzer
```

Zum besseren Verständnis der Arbeitsweise des Rasterimports ist im Folgenden ein gekürzter Auszug aus den generierten SQL-Anweisungen dargestellt. Die Einschübe … stehen dabei für längere Zahlen und Zeichenfolgen, die Geometrien oder die eigentlichen Raster WKB Daten definieren:

```
BEGIN;
CREATE TABLE "public"."landsat" (rid serial PRIMARY KEY, "filename" text);

SELECT AddRasterColumn('public','landsat','rast',25833,
ARRAY['8BUI','8BUI','8BUI','8BUI','8BUI','8BUI'], false, true, null, 28.5...,
-28.5..., 200, 200, ST_Envelope(ST_SetSRID('POLYGON((...))'::geometry, 25833)));

INSERT INTO "public"."landsat" ( filename, rast ) VALUES
( ('neubrandenburg.tif')::text, ('01000006...

CREATE INDEX "landsat_rast_gist_idx" ON "public"."landsat" USING GIST
(st_convexhull(rast));
END;
```

Der Auszug zeigt die grundlegenden Schritte des Ladevorgangs:

- Die Tabellendefinition;
- Der Aufruf von AddRasterColumn zur Erstellung und Registrierung der Rasterspalte;
- Einfügen der eigentlichen Rasterdaten durch mehrere INSERT Anweisungen;
- Erstellung eines Indizes auf die Ausdehnungen der Raster, sofern der -I Parameter übergeben wurde.

Erstellen eines Overviews für neubrandenburg.tif:

Aufgrund eines Fehlers in raster2pgsql.py, der zum Erstellungszeitpunkt noch nicht behoben wurde, müssen die generierten SQL Skripte manuell editiert werden, um korrekt zu funktionieren. [vgl 11]

Aus diesem Grund wird zunächst ein SQL Skript erstellt.

```
python /usr/lib/postgresql/8.4/bin/raster2pgsql.py -s 25833 -k 200x200  -r
neubrandenburg.tif -F -t public.landsat -l 2 -I > landsat_o2.sql
```

Der Aufbau der Overview Generierung unterscheidet sich nur in den Ersten zwei Schritten von der anderer Rastertabellen. Dieser Teil wird im Folgenden dargestellt:

```
BEGIN;

CREATE TABLE "public"."o_2_landsat" (rid serial PRIMARY KEY, "filename" text,
"rast" raster);

INSERT INTO public.raster_overviews( o_table_catalog, o_table_schema,
o_table_name, o_column, r_table_catalog, r_table_schema, r_table_name, r_column,
out_db, overview_factor) VALUES ('', 'public', 'o_2_landsat', 'rast', '',
'public', 'landsat', 'rast', FALSE, 2);
...
```

Die fett gedruckten Teile müssen ergänzt werden, damit das Skript in die Datenbank geladen werden kann.

Die Unterschiede zur normalen Rastererstellung bestehen darin, dass die Rasterspalte ohne den Aufruf von AddRasterColumn erzeugt wird und in der Eintragung der Metadaten in die RASTER_OVERVIEWS Tabelle.

Anschließend kann das Overview Skript in die Datenbank geladen werden:

```
psql -d postgis_tutorial -U pgnutzer -f landsat_o2.sql
```

Laden der Orthophotos:

Der Unterschied zum Laden des Landsat Bildes besteht darin, dass alle sechs Dateien in eine Tabelle geladen werden sollen.

Unter Nutzung von Platzhaltern sieht der Ablauf wie folgt aus:

```
cd /Pfad_zu_den_Bilddateien/

python /usr/lib/postgresql/8.4/bin/raster2pgsql.py -s 25833 -k 200x200 -r *.jpg
-F -t public.ortho -I | psql -d postgis_tutorial -U pgnutzer
```

Da die Nutzung von Platzhaltern auf dem Testsystem nicht möglich war, wurden die Dateien wie folgt geladen:

```
cd /Pfad_zu_den_Bilddateien/

python /usr/lib/postgresql/8.4/bin/raster2pgsql.py -c -s 25833 -k 200x200 -r
333825934.jpg -F -t public.ortho | psql -d postgis_tutorial -U pgnutzer

python /usr/lib/postgresql/8.4/bin/raster2pgsql.py -a -s 25833 -k 200x200  -r
333825936.jpg -F -t public.ortho | psql -d postgis_tutorial -U pgnutzer
...
```

Der Unterschied zur oberen Vorgehensweise besteht darin, dass zuerst das erste Bild geladen wurde, sodass

die Tabelle erstellt wurde. Danach wurden alle anderen Bilder wie im dritten Schritt dargestellt an diese Tabelle angefügt (Nutzung des -a Parameters). Aufgrund dieses Vorgehens konnte im Rahmen des Ladeprozesses kein Index erstellt werden. Dies wurde anschließend manuell durchgeführt.

4.2.2 Export von Rastern mithilfe des GDAL PostGIS Treibers

Die derzeit beste Möglichkeit, Rasterdaten aus PostGIS zu exportieren stellt das GDAL Toolkit dar. Dieses sollte im Rahmen der Installation in der Version 1.8 aus den Quellen übersetzt und installiert worden sein.

Für den Export aus PostGIS kann sowohl gdal_translate als auch gdalwarp verwendet werden. Beide Programme bieten zusätzlich weitere Bearbeitungsmöglichkeiten der Daten, auf die hier aber nicht eingegangen werden soll. Der Export über GDAL steht derzeit allerdings nur für regulär gekachelte Raster zur Verfügung. [vgl.06 S.398-400]

Ein spezielles PostGIS eigenes Werkzeug für den Export von Rastern, ähnlich pgsql2shp für Vektorgeometrien, ist vorerst nicht geplant.[vgl. 10]

Sowohl für QGIS als auch für gvSIG existieren Plugins für den Import von Rastern aus dem älteren WKTRaster Projekt. Das aktuelle PostGIS Raster wird allerdings noch nicht unterstützt.

Export mittels gdal_translate

```
gdal_translate -of GTIFF PG:"host=localhost dbname='postgis_tutorial'
user='pgnutzer' password='PostGISVM' schema='public' table='ortho' mode='2'"
ortho.tif
```

Die meisten verwendeten Optionen dienen der Anmeldung an der Datenbank und der Benennung der zu exportierenden Rasterspalte. Der Parameter „mode“ kann die Werte 1 und 2 annehmen. Der Wert 1 bewirkt, dass alle Kacheln in der Tabelle als eigenständige Raster exportiert werden, der Wert 2 führt zum Export aller Kacheln einer Tabelle zusammengefasst zu einer Bilddatei.

Der hier dargestellte Export erzeugt aus den sechs Orthophotos in der Tabelle ortho also ein einziges Bild. Dieser Vorgang kann aufgrund der großen Datenmenge je nach zu Verfügung stehender Hardware relativ lange dauern.

Abbildung 26: Stark verkleinerte Version der exportierten Orthophotodatei

4.3 Übersicht der PostGIS Raster Funktionen

Im Rahmen der Integration von WKTRaster in PostGIS-2.0 wurden weitere, rasterbezogene, Funktionen hinzugefügt. Eine Auflistung dieser Funktionen mit Verlinkung der genauen Beschreibungen und ggf. Anwendungsbeispielen steht in der offiziellen Dokumentation in Kapitel 8 zur Verfügung.

Die hier aufgelisteten Funktionen sind weitgehend aus diesem Kapitel entnommen worden.

Die Gruppierungen wurden im Rahmen dieses Tutorials in ähnlicher Weise wie unter 3.3 Übersicht der PostGIS Funktionen vorgenommen.

4.3.1 Rasterbezogene Verwaltungsfunktionen

Diese Gruppe enthält Funktionen zur Abfrage der Versionen der für Raster relevanten Komponenten.

PostGIS_Raster_Lib_Build_Date
PostGIS_Raster_Lib_Version
ST_GDALDrivers

4.3.2 Funktionen zum Umgang mit den Metadatentabellen

Die Funktionen dieser Gruppe dienen dem Umgang mit der RASTER_COLUMNS Metadatentabelle.

AddRasterColumn
DropRasterColumn
DropRasterTable

4.3.3 Ausgabefunktionen

Diese Gruppe enthält Funktionen zur Ausgabe von Rasterdaten in verschiedenen Formaten. Eine direkte Erzeugung von Rasterdateien durch Umleitung der Ausgabe dieser Funktionen war jedoch nicht möglich.

ST_AsBinary
ST_AsGDALRaster
ST_AsJPEG
ST_AsPNG
ST_AsTIFF

4.3.4 Funktionen mit Bezug auf Georeferenzierung und Bezugssysteme

Die Funktionen dieser Gruppe dienen dem Abfragen und Setzen von Metadaten mit Bezug zur Georeferenzierung sowie zur Abfrage und Transformation des Räumlichen Bezugssystems der Raster.

ST_GeoReference
ST_ScaleX
ST_ScaleY
ST_SkewX
ST_SkewY
ST_SRID
ST_UpperLeftX
ST_UpperLeftY

ST_SetGeoReference
ST_SetScale
ST_SetSkew
ST_SetSRID
ST_SetUpperLeft
ST_Transform

4.3.5 Funktionen zur Abfrage von Eigenschaften und Attributen

Die Funktionen dieser Gruppe dienen der Abfrage von Metadaten und Zuständen. Dabei wurden drei Untergruppen gebildet, die berücksichtigen, ob sich die abgefragten Werte auf ganze Raster, einzelne Bänder oder Pixel bezieht.

Rasterebene

ST_Height
ST_MetaData
ST_NumBands
ST_Width
ST_IsEmpty

Bandebene

ST_BandMetaData
ST_BandNoDataValue
ST_BandIsNoData
ST_BandPath
ST_BandPixelType
ST_HasNoBand

Pixelebene

ST_PixelAsPolygon
ST_Value
ST_Raster2WorldCoordX
ST_Raster2WorldCoordY
ST_World2RasterCoordX
ST_World2RasterCoordY

4.3.6 Funktionen zur Manipulation und Konstruktion von Rastern

Die Funktionen dieser Gruppe dienen der Veränderung bzw. der Erzeugung neuer Raster aufgrund von Ausgangsrastern.

ST_MakeEmptyRaster
ST_AddBand
ST_Band
ST_SetValue
ST_Transform
ST_SetBandNoDataValue
ST_SetBandIsNoData

ST_MapAlgebra .
ST_Reclass

4.3.7 Konvertierungsfunktionen

Die Funktion dieser Gruppe dienen der Konvertierung von Rastern in Polygone. Weitere Konvertierungsfunktionen, sowohl von Raster zu Raster, Raster zu Polygon als auch Polygon zu Raster, sind geplant oder befinden sich in der Entwicklung [vgl. 04]

ST_PixelAsPolygon
ST_Polygon
ST_DumpAsPolygons.

4.3.8 Verschneidungsfunktionen

Die Funktionen dieser Gruppe dienen der Verschneidung von Rastern und Geometriedaten. Weitere Funktionen in diesem Bereich sind geplant oder befinden sich in der Entwicklung. [vgl. 04]

ST_Intersection
ST_Intersects

4.3.9 Statistikenfunktionen

Die Funktionen dieser Gruppe dienen der Ermittlung verschiedener statistischer Daten des Rasters, insbesondere der Pixelwerte.

ST_Count
ST_Histogram
ST_Quantile
ST_SummaryStats
ST_ValueCount

4.3.10 Bounding Box Funktionen

Die Funktionen dieser Gruppe verhalten sich ähnlich wie ihr Gegenstück für Vektorgeometrien. Bei Rastern kann für jede Kachel eine Bounding Box erzeugt werden, sodass Analysefunktionen und Verschneidungen bei kleiner gekachelten Rastern aufgrund der besseren Nutzung der GIST Indizes wesentlich schneller ablaufen können.

Box2D
ST_ConvexHull
ST_Envelope

Räumliche Operatoren

&&
&<
&>

4.4 Beispielszenarien

Die folgenden Beispiele verwenden die „landsat“ Tabelle, die aus der Datei „neubrandenburg.tif“ erstellt wurde, als Grundlage. Dies liegt zum einen in den für die Beispielanalysen notwendigen spektralen Daten der Satellitenaufnahme begründet. Zum anderen ist dieser Datensatz wesentlich kleiner als die Orthophotos, was die notwendige Rechenzeit für Analysefunktionen und ggf. Export durch GDAL verkürzt.

4.4.1 Georeferenzierung und Transformation in andere Bezugssysteme

Die Georeferenzierungsmetadaten eines Rasters können über die Funktion `ST_GeoReference` abgefragt und mithilfe von `ST_SetGeoReference` (neu) festgelegt werden. Der Umgang mit Bezugssystemen erfolgt analog zu den Geometriedatentypen über `ST_SRID`, `ST_SetSRID` und `ST_Transform`. Dabei besitzt jedes Raster, also jede Kachel einer Rasterspalte eigene Referenzierungsdaten.

Georeferenzierung

Die Anzeige der Werte durch `ST_GeoReference` erfolgt nach dem von GDAL eingesetzten Schema:

scalex
skewy
skewx
scaley
upperleftx
upperlefty

Alternativ kann auch das ESRI Format ausgegeben werden.

Beispielabfrage:

```
SELECT rid, ST_SRID(rast), ST_GeoReference(rast)
FROM landsat;
```

rid	st_srid	st_georeference
1	25833	28.5000000000 0.0000000000 0.0000000000 -28.5000000000 371483.2500000000 5942064.7500000000
2	25833	28.5000000000 0.0000000000 0.0000000000 -28.5000000000 377183.2500000000 5942064.7500000000

An dieser Stelle sind aus Platzgründen nur die Ergebnisse der ersten beiden Kacheln dargestellt. Es ist ersichtlich, dass sich die Werte in der X-Koordinate der oberen linken Ecke unterscheiden, was aufgrund der unterschiedlichen räumlichen Position jeder Kachel auch notwendig ist. Die Georeferenzierungswerte werden beim Import der Raster durch raster2pgsql.py eingelesen bzw. für die Kacheln berechnet. Diese Metadaten sind auch in Rastern, die durch Analysefunktionen von den ursprünglich importierten Rastern abgeleitet wurden, enthalten.

Bei der Festlegung der Georeferenzierung durch `ST_SetGeoReference` muss berücksichtigt werden, dass die Koordinatenwerte für jede Kachel neu berechnet bzw. festgelegt werden müssen.

Beispielabfrage:

```
UPDATE landsat SET rast = ST_SetGeoReference(rast, '28.5 0 0 -28.5 371483.25
5942064.75','GDAL')
WHERE rid=1;
```

Neben den beiden `ST_GeoReference` und `ST_SetGeoReference` existieren Funktionen, die das Auslesen und Manipulieren der einzelnen Werte ermöglichen.

Transformation zwischen Bezugssystemen

Das Ziel des PostGIS Raster Projektes ist es, die kombinierte Nutzung von Geometrie und Rasterfunktion zu unterstützen und wo möglich äquivalente Funktionen für Raster bereit zu stellen. Im Falle der Bezugssystem-Funktionen können die Funktionen `ST_SRID`, `ST_SetSRID` und `ST_Transform` fast genauso genutzt werden wie bei Geometrien. Die grundlegenden Abläufe zur Erstellung neuer SRID bezogener Spalten oder Tabellen bzw. der direkten Nutzung von UPDATE Anweisungen unter temporärer Aufhebung von COSTRAINTs können hier ebenfalls angewendet werden.

`ST_Transform` unterscheidet sich für Raster dadurch, dass zusätzlich eine der von GDAL unterstützten Interpolationsmethoden übergeben werden muss.

Beispielabfrage:

```
CREATE TABLE landsat_krassowski AS

SELECT rid,ST_Transform(rast,28403,'NearestNeighbor') AS rast
FROM landsat
```

Hinweis:

Bei Versuchen mit den hier verwendeten Programmversionen sind bei der Transformation gekachelter Raster jedes mal erhebliche Bildartefakte in Form schwarzer Balken und Flächen entstanden. Nicht gekachelte Raster wurden korrekt transformiert. Eine Alternative zur Transformation innerhalb von PostGIS stellt der Export der Dateien mit anschließender Transformation durch gdalwarp und Reimport durch raster2pgsql.py dar.

4.4.2 Erstellung eigener Funktionen und Konstruktion abgeleiteter Raster

Fallbeispiel: Der Zustand der Wälder und sonstiger Vegetation in Neubrandenburg soll mithilfe des Landsat Bildes analysiert werden.

Zunächst wird für den besseren Überblick ein Bild erstellt, in dem die Vegetation deutlich zu unterscheiden ist. Dafür können die Bänder 4-3-2 bzw. nahes Infrarot-Rot-Grün respektive auf die Farbkanäle RGB abgebildet werden. GIS Software mit Rasterunterstützung bietet die Möglichkeit, die Kanäle auszuwählen und zu kombinieren, normale Bildbetrachtungssoftware stellt aber für gewöhnlich immer die ersten drei Bänder als Rot-Grün-Blau dar.

Im Folgenden wird ein Raster erstellt, dass nur die drei Bänder 4, 3 und 2 der Landsat Aufnahme hat:

```
CREATE TABLE landsat_veg AS

SELECT rid, ST_Band(rast,array[4,3,2]) AS rast
FROM landsat
```

`ST_Band` gibt ein neues Raster zurück, das aus den angegebenen Bändern des übergebenen Rasters besteht.

Ergebnis:

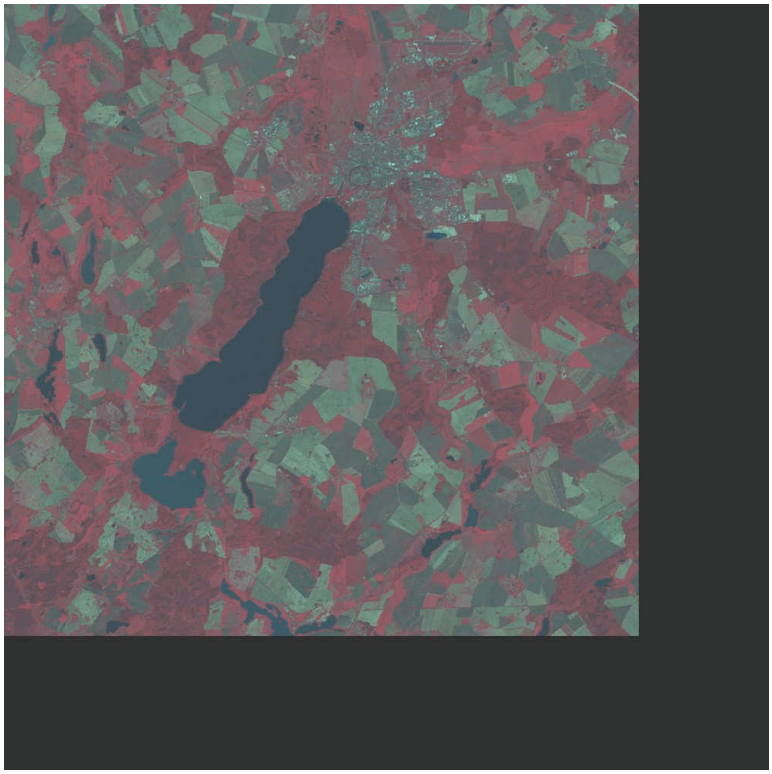

Abbildung 27: Abfrageergebnis - Raster mit den Bändern 4-3-2 der Landsat Aufnahme (verkleinert und leicht aufgehellt)

Die relativ breiten schwarzen Ränder am unteren und rechten Rand des Bildes ergeben sich aus der gewählten Kachelung von 200 mal 200 Pixeln.Die Teile der Kacheln, für die keine Bilddaten vorhanden sind, werden mit dem Wert Null (0, nicht NULL) aufgefüllt. Dies betrifft alle Bänder der betroffenen Kacheln.

Berechnung des NDVI

Fallbeispiel:

Zur besseren Analyse des Zustandes der Vegetation soll der „Normalized Difference Vegation Index“ oder NDVI für die Region berechnet werden. Dieser Index beruht auf der Tatsache, dass gesunde Vegetation rotes Licht gut absorbiert, aber infrarote Wellenlängen stärker reflektiert. Der Index ist ein Wert zwischen -1 und 1, wobei Werte nahe an 1 für gesunde Vegetation kennzeichnende sind.

PostGIS Raster stellt die Funktion `ST_MapAlgebra` zur Verfügung, die ermöglicht, auf die Pixel eines Bandes eines Rasters jeden, in PostgreSQL gültigen arithmetischen Ausdruck anzuwenden. Sie kann für alle Bildmanipulationen, die nur Daten eines Bandes benötigen, angewandt werden.

Die Berechnung des NDVI erfordert jedoch den Zugriff auf zwei Bänder bei jeder Pixelberechnung. Da zum Erstellungszeitpunkt noch keine entsprechende Version `ST_MapAlgebra` vorliegt, wurde die folgende, prototypische Funktion für die Berechnung des NDVI erstellt. PostgreSQL ermöglicht die Erstellung von pl/pgsql Funktionen über normale Datenbank Clients. In diesem Fall wurde pgadmin3 eingesetzt, die Verwendung von psql ist aber genauso möglich.

Anweisung zur Erstellung der pl/pgsql Funktion:

```
CREATE OR REPLACE FUNCTION tuto_NDVI(Ausgangsraster raster, IR_band int,
RED_band int)
  RETURNS raster AS $$
  DECLARE
    NDVIrast raster := ST_MakeEmptyRaster(Ausgangsraster);
    Bildhoehe int:=ST_Height(Ausgangsraster);
    Bildbreite int:=ST_Width(Ausgangsraster);
    IR_Wert double precision;
    RED_Wert double precision;
    NDVI_Pixelwert double precision;
 BEGIN
    NDVIrast := ST_AddBand(NDVIrast,'64BF');
    FOR rowy IN 1..Bildhoehe LOOP
        FOR columnx IN 1..Bildbreite LOOP
            IR_Wert:=ST_Value(Ausgangsraster,IR_band,columnx,rowy);
            RED_Wert:=ST_Value(Ausgangsraster,RED_band,columnx,rowy);
            IF (IR_Wert+RED_Wert)=0 THEN
                 NDVI_Pixelwert:=0;
            ELSE NDVI_Pixelwert:=((IR_Wert-RED_Wert)/(IR_Wert+RED_Wert));
            END IF;
          NDVIrast:=ST_SetValue(NDVIrast,columnx,rowy,NDVI_Pixelwert);
      END LOOP;
    END LOOP;
  RETURN NDVIrast;
  END;
  $$ LANGUAGE 'plpgsql';
```

Wie in `tuto_NDVI` ersichtlich, können alle PostGIS Funktionen, die in der Datenbank geladen sind, von selbst geschriebenen Funktionen genutzt werden.

`ST_MakeEmptyRaster` wird hier eingesetzt, um ein leeres Ausgangsraster zu erzeugen, das dieselben Metadaten und dieselbe Kachelung wie das Ausgangsraster besitzt, jedoch keine Bänder enthält.

`ST_AddBand` wird verwendet, um dem neuen Raster ein Band mit dem Pixeltyp 64BF, also Fließkommawerten mit doppelter Genauigkeit, hinzuzufügen. Die Funktion kann in anderen Zusammenhängen auch eingesetzt werden, um Raster zu erstellen, die Bänder aus unterschiedlichen Ausgangsrastern enthalten.

Die Verzweigung im Rumpf der inneren Schleife ist nötig, um die erwähnten Nullwerte an den Rändern zu behandeln.

Erstellung des NDVI Rasters:

```
CREATE TABLE ndvi AS

SELECT rid, tuto_NDVI(rast,4,3) AS rast
FROM landsat
```

Hinweis: Die Funktion benötigt relativ viel Rechenzeit, was natürlich auch Hardware abhängig ist. Auf dem Testsystem betrug die Ausführungszeit ca. 35 Minuten.

Ergebnis:

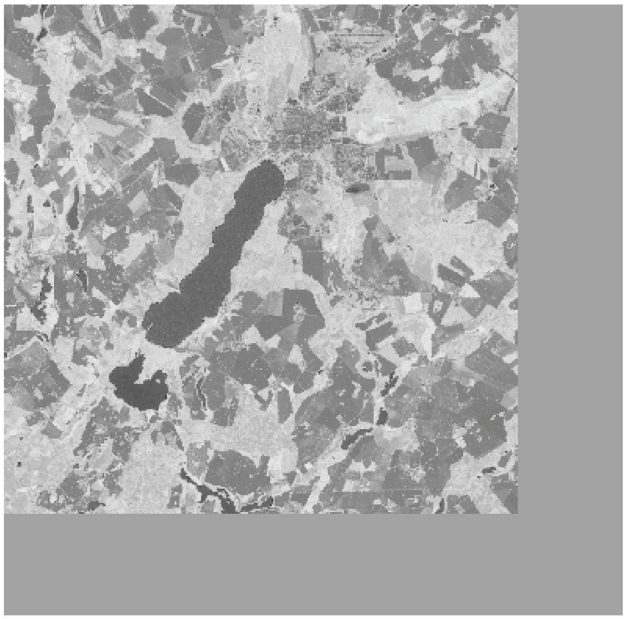

Abbildung 28: Abfrageergebnis - Darstellung des NDVI-Rasters in gvSIG (verkleinert)

Das Resultierende Raster stellt die Werte nahe 1 hell und die nahe -1 dunkel dar. Für die korrekte Darstellung wird jedoch eine geeignete Software wie gvSIG benötigt.

4.4.3 Reklassifizierung

Fallbeispiel:

Zur besseren Weiterverarbeitung soll eine thematische Karte erstellt werden, die alle Bereiche mit und ohne Vegetation deutlich unterscheidet. Der Einfachheit halber soll eine binäre Karte erstellt werden, die alle Gebiete mit Vegetation schwarz und alle anderen weiß darstellt.

Abfrage:

```
CREATE TABLE ndvi_reclass AS

SELECT rid,ST_Reclass(rast,'-1-0.2:1,0.2-1:0','1BB') AS rast
FROM ndvi
```

Die Funktion `ST_Reclass` erlaubt es, Ausgangswertebereiche und die dazugehörigen Ersatzwerte zu definieren. Der letzte Parameter bestimmt den Pixeltyp des neuen Rasters.

Ergebnis:

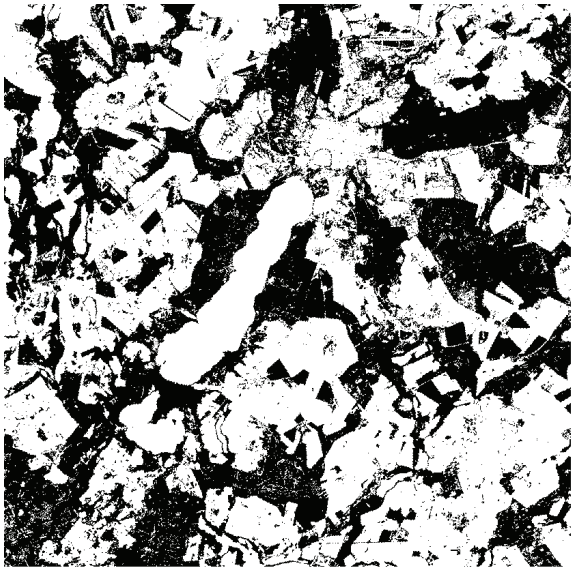

Abbildung 29: Abfrageergebnis - Reklassifiziertes binäres Raster mit Darstellung von Gebieten mit gesunder Vegetation (verkleinert)

4.4.4 Polygonisierung

Die PostGIS Raster Funktionen zur Konvertierung von Raster- in Vektordaten basieren auf der Zusammenfassung von Pixeln mit gleichen Werten. Daher sind thematische und reklassifizierte Karten mit wenigen Pixelwerten in zusammenhängenden Flächen geeignet. Bilder mit großer Farbvielfalt, wie etwa Luftbilder, haben kaum zusammenhängende Flächen mit gleichen Pixelwerten, was zur Folge hat, dass nahezu die gleiche Anzahl von Polygonen wie die Zahl der Pixel erzeugt wird. Eine Polygonisierung würde in diesem Fall sehr lange dauern, große Datenmengen erzeugen und kaum brauchbare Ergebnisse liefern.

Für die Polygonisierung von Flächen stehen die Funktionen `ST_Polygon` und `ST_DumpAsPolygons` zur Verfügung.

`ST_Polygon` erzeugt aus allen Pixeln, deren Wert nicht dem NoData Wert entspricht, eine Sammlung von Polygonen. Ist der NoData Wert NULL, was den Ausgangswert für neue Raster darstellt, liefert `ST_Polygon` immer ein Polygon mit den Maßen des Rasters zurück.

`ST_DumpAsPolygons` erzeugt für alle im Raster vorkommenden Pixelwerte sogenannte `geomval` Objekte. Diese Objekte enthalten Geometrien und die zu diesen Geometrien gehörenden Pixelwerte Dabei werden zusammenhängende Flächen mit gleichen Werten zu zusammenhängenden Polygonen zusammengefasst.

Die Funktion ist auf diese Weise geeignet, thematische Rasterkarten zu konvertieren, da auch die zu den Flächen gehörenden Pixelwerte gespeichert werden. Diesen können dann mithilfe einer Legende Attribute zugeordnet werden.

`ST_DumpAsPolygons` berücksichtigt NoData Werte und erzeugt keine Objekte für Pixel dieses Wertes.

Fallbeispiel: Aus der binären Karte sollen Vektorielle Repräsentationen der Vegetationsgebiete entwickelt werden.

Beispiel unter Nutzung von ST_Polygon:

```
CREATE SEQUENCE "testseq";
ALTER TABLE "testseq" OWNER TO pgnutzer;


UPDATE ndvi_reclass
      SET rast = ST_SetBandNoDataValue(rast,1,1)


CREATE TABLE ndvi_poly2 AS

SELECT nextval('testseq')::int4 AS gid, ST_Polygon(rast) AS geom
FROM ndvi_reclass
```

`ST_SetBandNoDataValue` dient zur Festlegung des Nodata Wertes, ohne den `ST_Polygon` nicht sinnvoll eingesetzt werden kann. Der Wert 1 stellt in `ndvi_reclass` alle Gebiete dar, die keine Vegetation darstellen.

Die Erstellung der Sequenz „testseq“ und die Anweisung zur Erzeugung des Feldes „gid“ dienen dem Zweck, die neu erstellte Tabelle kompatibler mit GIS Desktopanwendungen zu gestalten.

Beispiel unter Nutzung von ST_DumpAsPolygons:

```
CREATE SEQUENCE "testseq";
ALTER TABLE "Test" OWNER TO pgnutzer;

UPDATE ndvi_reclass
      SET rast = ST_SetBandNoDataValue(rast,1,1)

CREATE TABLE ndvi_poly AS

SELECT nextval('testseq')::int4 AS gid,(ST_DumpAsPolygons(rast)).val AS
Klassifikation,(ST_DumpAsPolygons(rast)).geom AS geom
FROM ndvi_reclass
```

Ergebnis:

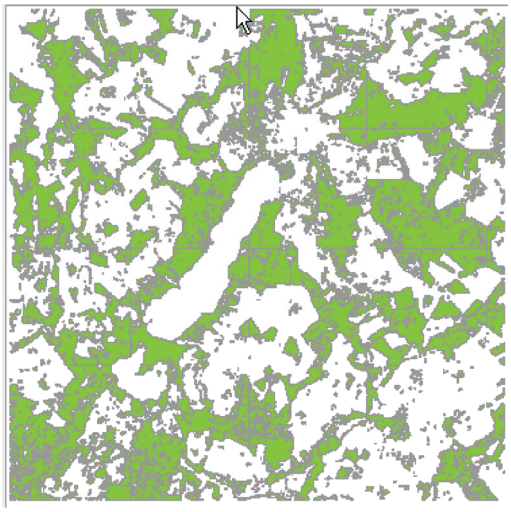

Abbildung 30: Abfrageergebnis - Darstellung des erstellten Polygon-Layers der Vegetationsgebiete

Unter den gegebenen Bedingungen erzeugen beide Funktionen nahezu das gleiche Ergebnis mit der Ausnahme, dass durch `ST_DumpAsPolygons` zusätzlich das Klassifikationsfeld erzeugt werden kann. Dies liegt darin begründet, dass `ndvi_reclass` eine binäre Karte ist, in der außer dem NoData Wert keine anderen Pixelwerte vorkommen. Ist der NoData Wert hingegen NULL erzeugt `ST_DumpAsPolygons` zwei Gruppierungen.

Alternatves Beispiel für ST_DumpAsPolygons:

```
UPDATE ndvi_reclass
      SET rast = ST_SetBandNoDataValue(rast,1,NULL)

CREATE TABLE ndvi_poly AS

SELECT nextval('testseq')::int4 AS gid,(ST_DumpAsPolygons(rast)).val AS
Klassifikation,(ST_DumpAsPolygons(rast)).geom AS geom
FROM ndvi_reclass
```

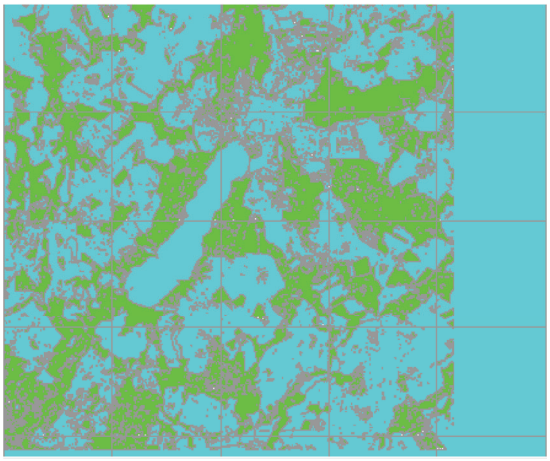

Abbildung 31: Abfrageergebnis - Darstellung der Polygonisierung der Vegetationsgebiete in grün und der anderen Bereiche in blau.

4.4.5 Verschneidung

Die Funktion `ST_Intersection` ermöglicht die Verschneidung von Rastern und Geometrien. Dabei wird der von der Verschneidung betroffene Bereich zunächst vektorisiert, sodass wie bei `ST_DumpAsPolygons geomval` Objekte für alle betroffenen Pixelwerte entstehen und dann mit der Geometrie verschnitten.

`ST_Intersects` erfüllt dieselbe Funktion wie bei Geometrien. Die Funktion prüft auf Überschneidung zwischen der Geometrie und der konvexen Hüllen der Rasterkacheln, die ebenfalls Polygone darstellen.

Fallbeispiel: Es soll eine Verschneidung des Auswirkungsbereichs der geplanten Ortsumgehung um Neubrandenburg mit der in `ndvi_reclass` erzeugten Rasterkarte der Vegetation in diesem Gebiet erstellt werden, um die schädlichen Auswirkungen auf die Natur in weiteren Schritten besser analysieren zu können.

```
CREATE TABLE betroffene_vegetation AS

SELECT nextval('testseq')::int4 AS gid,(ST_Intersection(geom,rast)).val AS
Klassifikation, (ST_Intersection(geom,rast)).geom AS geom
FROM umgehungbuffer, ndvi_reclass
WHERE Intersects(rast,geom);
```

Ergebnis:

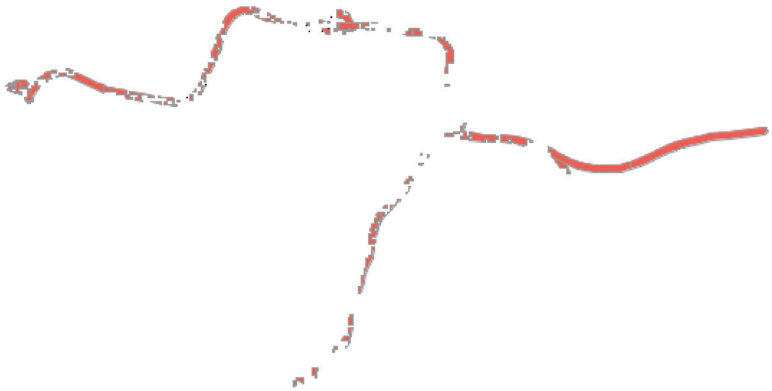

Abbildung 32: Abfrageergebnis - Vektorisierung der Überschneidung der Vegetationsgebiete mit der Ortsumgehung

Abbildung 33: Gemeinsame Darstellung der Vektorisierung des Verschneidungsgebietes und der Vektorisierung des Vegetationsgebietes

5 Zusammenfassung und Ausblick

PostGIS bietet die Möglichkeit, Geographische Daten in einer PostgreSQL Datenbank zu speichern. Damit ermöglicht es zum einen, alle Vorteile der Datenbank basierten Speicherung gegenüber Datei basierter Speicherung zu nutzen, und zum anderen, die GIS Daten mit anderen Datensätzen, die schon länger in einer Datenbank gespeichert wurden, zu verknüpfen.

Die von PostGIS angebotenen Funktionen zur Analyse und Verarbeitung von Geodaten decken aktuell einen großen Teil der Aufgabenbereiche von Desktop GIS Anwendungen ab. Gleichzeitig existieren in den meisten Desktop GIS Programmen Interfaces für die Anbindung von PostGIS Datenbanken als Datenquellen.

Dabei ermöglicht der Ansatz, Analysen in der Geodatenbank auszuführen, die Automatisierung und Vereinfachung von Arbeitsabläufen, da kein Import und Export zu anderen Anwendungen nötig ist. Dabei ist die SQL basierte Methode zur Erstellung von Analysen besonders geeignet, um im Vorhinein definierte Anweisungen auf große Datenbestände anzuwenden.

Ein weiteres Anwendungsgebiet sind Thin Client und Webanwendungen, die im Zusammenspiel mit PostGIS nur die Nutzerschnittstelle darstellen müssen, da Speicherung und Analyse in der Datenbank stattfinden können.

Die Unterstützung von Kurvenobjekten und 3D Geometrien befindet sich noch in einem relativ frühen Entwicklungsstadium. Langfristig eröffnet sich aber die Perspektive, anspruchsvollere 3D Anwendungen wie Stadtmodelle über PostGIS zu realisieren. Bei entsprechender Weiterentwicklung dieser Funktionen besteht die Möglichkeit, dass auch andere Anwendungsbereiche wie CAD oder 3D-Design im Allgemeinen PostGIS als Mittel zur Datenspeicherung und Verarbeitung einsetzen werden.

Angesichts dieser bereits bestehenden Möglichkeiten sowie der Entwicklungspotentiale kann man berechtigterweise davon ausgehen, dass PostGIS im Bereich freier und GIS Software im Allgemeinen weiter an Bedeutung gewinnen wird.

Ziel dieser Arbeit war es, in der zur Verfügung stehenden Bearbeitungszeit einen möglichst umfangreichen und gut verständlichen Einstieg in diese Technologie zu bieten. Dabei muss allerdings schon allein aufgrund der Ausrichtung auf eine Entwicklungsversion der Software davon ausgegangen werden, dass sich Teile der dargestellten Funktionsweisen und Strukturen in relativ kurzer Zeit ändern werden.

Quellenverzeichnis

[01] offizielle Dokumentation der PostGIS 2.0 Entwicklungsversion rev 7527
URL: http://postgis.refractions.net/documentation/manual-svn/
(30.06.2011)

[02] Darstellung der Entwicklungsgeschichte von PostgreSQL
URL: http://www.postgresql.org/about/history
(15.07.2011)

[03] Darstellung der Entwicklungsgeschichte von PostGIS
URL: http://www.refractions.net/products/postgis/history/
(15.07.2011)

[04] Wiki des WKTRaster Projekts
URL: http://trac.osgeo.org/postgis/wiki/WKTRaster
(15.07.2011)

[05] offizielle Dokumentation des PostgreSQL Projektes
URL:http://www.postgresql.org/docs/8.4/interactive/index.html
(11.07.2011)

[06] Obe, Regina O./ Hsu, Leo S. (2011) : PostGIS in Action, Manning Publications Co.

[07] Diskussion der Umwandlung der GEOMETRY_COLUMNS Tabelle im Taskmanagement System des PostGIS Projekts
URL:http://trac.osgeo.org/postgis/ticket/944
(09.08.2011)

[8] Diskussion in der PostGIS Mailing List zu 3D Typen
URL: http://postgis.refractions.net/pipermail/postgis-devel/2010-August/009841.html
(19.08.2011)

[9] Präsentationsfolien einer Präsentation auf der North Carolina GIS Conference 2011 über die Raster- und 3D Unterstützung in PostGIS-2.0 von Regina Obe und Leo Hsu.
URL: http://www.postgis.us/downloads/ncgis2011/NCGISSDBPostGIS20_2011.pdf
(02.08.2011)

[10] Diskussion in der PostGIS Mailing List PostGIS Raster Import und Export
URL: http://www.mail-archive.com/postgisusers@postgis.refractions.net/msg12093.html
(19.08.2011)

[11] Fehlermeldung zu raster2pgsql.py im Taskmanagement System des PostGIS Projekts
URL:http://trac.osgeo.org/postgis/ticket/825
(18.08.2011)

Abbildungsverzeichnis

Anhang

1 CD mit Inhalt:

- Kopie der Bachelorarbeit im PDF Format
- Die verwendeten Demodatensätze
- Die Quelltexte der verwendeten Versionen von PostGIS, GEOS und GDAL als Archivdateien