PostGIS - Freie Geodaten in Freien Datenbanken

Jens Pönisch

22. November 2010

1 Überblick

1.1 Aufgabenstellung

- Verwalten von geometrischen Objekten (Punkte, Linien, Flächen) in Datenbanken
- Aussagen über die Objekte: Größe, Lage
- Beziehung zwischen den Objekten: Berührung, Schnittpunkte, Teilmengen, Überdeckungen
- Verwenden verschiedener geographischer Bezugssysteme (*Spatial Reference Systems* wie Gauß-Krüger, WGS84, UTM), Umrechnungen
- Abstände und Flächen
- Problematik kam aus der Geoinformatik, Verwendung aber auch für andere geometrische Daten in karthesischen Koordinaten (2D und 3D) sinnvoll

1.2 Warum relationale Datenbank nicht ausreichend?

Beispiel Linie:

```
1 create table line (
2   id serial,
3   name varchar(20),
4   x0 int, y0 int,
5   x1 int, y1 int,
6   primary key (id)
7 );
```

- Test, ob zwei Linien identisch, ist schon schwierig (Richtung)
- Geometrische Funktionen wie Abstand, Schnittpunkt, Bounding-Box fehlen
- Wie Polygone speichern (unterschiedliche Eckenzahl)?

1.3 Erweiterung

- PostgreSQL unterstützt bereits Geomtrietypen (in den meisten Büchern nicht erwähnt)
- Für geographische Daten nicht ausreichend, nicht konform zur OpenGIS-Spezifikation
- Deshalb seit 2000 Entwicklung einer Erweiterung von *Refractions Research*: *PostGIS*
- Komplette Unterstützung der Basisfunktionalität (*Simple Features*)
- Inzwischen ist SQL/MM (weitere Datentypen, Multimedia) teilweise unterstützt.
- Deutlich mehr geometrische Funktionen als für die eingebauten Geometrietypen
- Unterstützung verschiedener geographischer SRS und Umrechnung mittels *Proj4*-Bibliothek

1.4 Umsetzung

Die Geometrieinformationen werden wie folgt bereit gestellt:

- Jede Tabelle kann *höchstens eine* Geometriespalte erhalten.
- Die Geometriespalte enthält geometrische Objekte samt Koordinaten eines geometrischen Typs (Punkt, Linienzug) oder von gemischten Typen.
- Die Geometriespalte wird mit den üblichen SQL-Mitteln abgefragt.
- Die Geometriedaten können als Text (WKT) oder in einem definierten Binärformat vorliegen.
- Geometrische Operationen werden durch eine Vielzahl von zusätzlichen Funktionen bereit gestellt.

1.5 Einsatz von PostGIS

- OSM-Renderer Mapnik nutzt PostGIS als Datenquelle.
- Inzwischen wurde die zentrale OSM-Datenbank von mySQL auf PostGIS umgestellt (wobei die PostGIS-Fähigkeiten nur teilweise genutzt werden).
- WMS-Server können PostGIS als Datenquelle nutzen.
- Viele GIS-Programme können PostGIS als Datenquelle nutzen.

2 Installation

2.1 Paketinstallation

- PostgreSQL installieren
- Debian: `postgresql-contrib` installieren
- Anschließend PostGIS installieren
- Eventuell Erweiterungen installieren

Installiert werden im wesentlichen einige *shared libraries* sowie SQL-Scripts.

2.2 PostGIS installieren

PostGIS muss in jeder Datenbank extra installiert werden. Entweder Template erstellen oder für jede Geo-Datenbank folgende Schritte durchführen:

- Serversprache `plpgsql` installieren
- PostGIS-Nutzer anlegen
- PostGIS-SQL-Scripts einspielen
- Für Verwendung mit OpenStreetMap Datei `900913.sql` einspielen

Unter Debian-Linux:

```
1  su - postgres
2  createdb -O owner -E utf-8 dbname
3  createlang plpgsql dbname
4  psql -d dbname -f /usr/share/postgresql-8.3-postgis/lwpostgis.sql
5  psql -d dbname -f /usr/share/postgresql-8.3-postgis/spatial_ref_sys.sql
6  wget http://trac.openstreetmap.org/browser/applications/utils/export/osm2pgsql/900913.sql?format=raw
7  psql -d dbname -f ./900913.sql
8  psql dbname
9  # alter table geometry_columns owner to owner;
10 # alter table spatial_ref_sys owner to owner;
```

In `pg_hba.conf` Zugriffsrechte auf die neue Datenbank eintragen.

2.3 PostGIS-Metatabellen

PostGIS installiert zwei Tabellen für Metadaten:

geometry_columns Metadaten der Geometriespalte einer Tabelle: SRS, Geometrietyp, ...
spatial_ref_sys Metadaten der SRS. Werden für Koordinatentransformationen benötigt.

2.4 Test

Prüfen der PostGIS-Funktionalität und der korrekten Koordinatentransformationen.

```
-- WGS84 -> Spherical Mercator
select st_astext(
  st_transform(
    st_geometryfromtext('POINT (12.9 50.8)', 4326),
    900913
  )
);
```

Ergebnis:

```
                st_astext
-----------------------------------------
 POINT(1436021.43123323 6585991.99809962)
```

Rücktransformation:

```
select st_astext(
  st_transform(
    st_geometryfromtext('POINT (1436021.43123323 6585991.99809962)', 900913),
    4326
  )
);
```

Ergebnis:

```
    st_astext
------------------
 POINT(12.9 50.8)
```

Falls Warnungen erscheinen und das Ergebnis der zweiten Transformation Unsinn ist, muss das Paket `proj` aktualisiert werden (Debian Lenny: aus Backports holen).

2.5 HStore

Seit kurzem benutzt OSM HStore-Felder (assoziative Arrays) für das Speichern von Tags.

Installation in Debian:

```
apt-get install postgresql-server-dev-8.3
cvs -d :pserver:anonymous@cvs.pgfoundry.org:/cvsroot/hstore-new checkout hstore-new
cd hstore-new
make
make install
```

Anschließend benötigte SQL-Scripte in alle OSM-Datenbanken einspielen:

```
psql dbname -f /usr/share/postgresql/8.3/contrib/hstore-new.sql
```

3 Dateneingabe

3.1 Beispiel

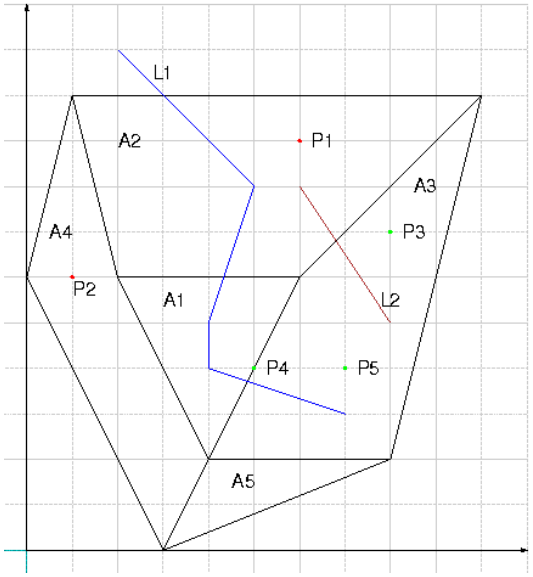

Abbildung 3.1: Einfache Geometrie

Tabellendefinition (`mixed.sql`)

3.2 Geometrietypen

PostGIS unterstützt folgende Geometrietypen:

Typ	Erklärung	Syntax
`POINT`	Punkt (x-, y-Koordinate)	`POINT(x y)`
`LINESTRING`	Linienzug (Folge von x-, y-Koordinaten)	`LINESTRING(x1 y1, x2 y2[,...])`
`POLYGON`	Polygon (Folgen von x-, y-Koordinaten, geschlossen) Endpunkt muss mit Anfangspunkt übereinstimmen	`POLYGON((x1 y1, x2 y2[,...], x1 y1) [,(punktliste)...])`
`MULTIPOINT`	Punktmenge	`MULTIPOINT(x1 y1, x2 y2[, ...])`
`MULTILINESTRING`	Menge von Linienzügen	`MULTILINESTRING((x1 y1, x2 y2[, ...]), (...))`
`MULTIPOLYGON`	Menge von Polygonen	`MULTIPOLYGON(((polygon1)), ((...)))`
`GEOMETRYCOLLECTION`	Menge verschiedener geometrischer Objekte	`GEOMETRYCOLLECTION(geometrie1, ...)`

Anmerkung 3.1. *Die Doppelklammern bei `POLYGON` schließen eine* Liste *von geschlossenen Linienzügen ein: Das Polygon darf Löcher enthalten.*

Weitere Typen aus SQL/MM werden ebenfalls unterstützt, sollen aber nicht behandelt werden.

3.3 Tabellen erzeugen

```
1 create table tabname ...;
2 select AddGeometryColumn('tabname', 'spaltenname', SRID, 'typ', dimension);
```

- Nur eine Geometriespalte pro Tabelle möglich
- `AddGeometryColumn()` erzeugt Metadateneintrag in Tabelle `geometry_columns`
- Ohne geographischen Bezug SRID -1 verwenden
- Typ ist einer der geometrischen Datentypen
- Für geometrische Daten verschiedener Typen wird der Typ `GEOMETRY` verwendet.

Beispiel:

```
1 create table mixed (
2   id serial,
3   name varchar(20),
4   color varchar(20),
5   primary key (id)
6 );
7 select AddGeometryColumn('mixed', 'geom', -1, 'GEOMETRY', 2);
```

3.4 Dateneingabe

Manuelle Eingabe als WKT -> Konvertierung erforderlich:

```
st_geomFromText('geometrieobjekt'[, srid])
```

`srid` kann entfallen (Default: -1).

Einfügen von Zeilen wie üblich mit `INSERT`:

```
insert into tabname (..., geomspalte)
  values (..., ST_GeomFromText('geometrieobjekt', srid));
```

Beispiel:

```
insert into mixed (name,geom) values ('A1', st_geomfromtext(
  'POLYGON ((4 2, 6 6, 2 6, 4 2))', -1)
);
insert into mixed (name,geom) values ('P1', st_geomfromtext(
  'POINT(6 9)', -1)
);
-- alternativ mit altem Funktionsnamen:
insert into mixed (name,geom) values ('L1', geometryfromtext(
  'LINESTRING(2 11, 5 8, 4 5, 4 4, 7 3)', -1)
);
```

3.5 Update und Löschen

- SQL-Anweisung `update` bzw. `delete`
- Konvertierung der Geometriedaten wie bei der Eingabe.
- Geometrie als Auswahlbedingung problemlos möglich.

```
update mixed set geom=st_geomfromtext('POINT(7 10)') where name='PX';
-- Achtung: Löscht alle Objekte mit gleicher Bounding-Box!
delete from mixed where geom=st_geomfromtext('POINT(7 10)');
-- Besser:
delete from mixed where st_equals(geom, st_geomfromtext('POINT(7 10)'));
```

4 Geometrische Abfragen

4.1 Geometriedaten

Die Geometriespalte kann einfach abgefragt werden:

```
select geometriespalte ... from tabelle ....;
```

Beispiel:

```
select geom from mixed where name='P1';
```

Ergebnis:

```
                    geom
-------------------------------------------
 0101000000000000000000184000000000000002240
```

Ausgabe erfolgt als hexadezimal codierter String des WKB-Formats.

Für interaktive Nutzung Decodierung ins WKT-Format erforderlich:

```
select st_astext(geometriespalte) ... from tabelle;
-- oder:
select st_asewkt(geometriespalte) ... from tabelle;
```

Beispiel:

```
select st_asewkt(geom) from mixed where name='P1';
```

Ergebnis:

```
 st_asewkt
------------
 POINT(6 9)
```

Anmerkung 4.1. *Haben unsere Geometriedaten ein SRS, unterscheiden sich die Ausgaben von* `st_astext()` *und* `st_asewkt()`*. Die zweite Form gibt das Bezugssystem mit aus.*

Binärdarstellung zur Weiterverarbeitung:

```
select st_asbinary(geometriespalte) ... from tabelle;
-- oder:
select st_asewkb(geometriespalte) ... from tabelle;
```

Beschreibung des Binärformats in [3]

4.2 Metadaten eines Geometrieobjekts

Funktion	Erklärung
`st_geometrytype(geometrie)`	Geometrietyp
`geometrytype(geometrie)`	Geometrietyp (andere Darstellung)
`st_ndims(geometrie)`	Anzahl Dimensionen
`st_srid(geometrie)`	SRS des Objekts

Abfrage Einzelobjekt:

```
select st_geometrytype(geom),st_ndims(geom),st_srid(geom)
  from st_geomfromtext('POINT(12.9 52.8)', 4326) as geom;
```

Ergebnis:

```
 st_geometrytype | st_ndims | st_srid
-----------------+----------+---------
 ST_Point        |        2 |    4326
```

Abfrage Tabelle:

```
select name,st_geometrytype(geom),st_ndims(geom),st_srid(geom) from mixed;
```

4.3 Abfragen für Punkte

Ein Punkt ist durch Angabe seiner Koordinaten vollständig charakterisiert.

Funktion	Erklärung
`st_x(punkt), st_y(punkt)`	x-, y-Koordinate eines Punkts

Beispiel:

```
select st_x(geom),st_y(geom)
  from st_geomfromtext('POINT(12.9 52.8)', 4326) as geom;
```

Ergebnis:

```
 st_x | st_y
------+------
 12.9 | 52.8
```

Bei Abfrage der gemischten Geometriespalte muss gefiltert werden:

```
select st_x(geom),st_y(geom) from mixed where st_geometrytype(geom)='ST_Point';
```

4.4 Abfragen für Linestrings

Funktion	Erklärung
`st_length(linestring)`	Gesamtlänge des Linestrings (Euklidische Norm)
`st_npoints(linestring)`	Anzahl der Stützstellen
`st_startpoint(linestring)`	Anfangspunkt
`st_endpoint(linestring)`	Endpunkt
`st_pointn(linestring, n)`	n-ter Punkt (Numerierung ab 1)
`st_closed(linestring)`	Prüfung auf geschlossenen Linienzug (erster gleich letzter Punkt)
`st_isring(linestring)`	Prüfung auf Ring (erster gleich letzter Punkt, keine Überschneidungen)

Beispiel:

```
select name,st_length(geom),st_NPoints(geom), st_astext(geom)
  from mixed
  where st_geometrytype(geom)='ST_LineString';
```

Ergebnis:

```
 name |    st_length     | st_npoints |            st_astext
------+------------------+------------+---------------------------------
 L1   |  11.567196007456 |          5 | LINESTRING(2 11,5 8,4 5,4 4,7 3)
 L2   | 3.60555127546399 |          2 | LINESTRING(6 8,8 5)
```

Punkte abfragen:

```
select st_astext(st_startpoint(geom)) as start,
       st_astext(st_endpoint(geom)) as end,
       st_astext(st_pointn(geom,2)) as point,
       st_astext(geom)
  from mixed where name='L1';
```

Ergebnis:

```
    start    |    end     |   point    |            st_astext
-------------+------------+------------+---------------------------------
 POINT(2 11) | POINT(7 3) | POINT(5 8) | LINESTRING(2 11,5 8,4 5,4 4,7 3)
```

4.5 Abfragen für Polygone

Funktion	Erklärung
`st_perimeter(polygon)`	Umfang des Polygons (alle Ringe, Euklidische Norm) `st_length()` liefert hier 0!
`st_npoints(polygon)`	Anzahl Punkte (= Ecken +1)
`st_area(polygon)`	Fläche (Löcher werden abgezogen)
`st_exteriorring(polygon)`	Äußerer Ring des Polygons als Linestring
`st_numinteriorrings(polygon)`	Anzahl der Löcher (inneren Ringe) des Polygons
`st_interiorring(polygon, nr)`	Spezifizierter innerer Ring des Polygons als Linestring

Beispiel:

```
select name,
       st_perimeter(geom),
       st_NPoints(geom),
       st_area(geom),
       st_astext(geom) from mixed
  where st_geometrytype(geom)='ST_Polygon';
```

Ergebnis:

```
 name |   st_perimeter   | st_npoints | st_area |              st_astext
------+------------------+------------+---------+-----------------------------------
 A1   | 12.9442719099992 |          4 |       8 | POLYGON((4 2,6 6,2 6,4 2))
 A2   |   22.77995987511 |          5 |      26 | POLYGON((2 6,6 6,10 10,1 10,2 6))
 A3   | 22.3752014557273 |          5 |      20 | POLYGON((4 2,8 2,10 10,6 6,4 2))
 A4   | 21.6626191162341 |          6 |      14 | POLYGON((0 6,3 0,4 2,2 6,1 10,0 6))
 A5   | 11.6212327846343 |          4 |       4 | POLYGON((3 0,8 2,4 2,3 0))
```

Äußerer Umfang eines Polygons:

```
select st_length(st_exteriorring(geom)) as outer_perimeter,
       st_astext(geom)
  from mixed where name='A1';
```

Ergebnis:

```
 outer_perimeter  |         st_astext
------------------+---------------------------
 12.9442719099992 | POLYGON((4 2,6 6,2 6,4 2))
```

5 Beziehungen zwischen Objekten

5.1 Gleichheit

Funktion	Erklärung
`st_equals(geometrie1, geometrie2)`	Geometrien sind gleich, i.e. beinhalten die gleichen Punkte
`st_disjoint(geometrie1, geometrie2)`	Geometrien sind disjunkt

Vergleich mit = vergleicht die Bounding-Box und funktioniert hier nicht wie erwünscht:

```
select st_geomfromtext('LINESTRING(1 0, 1 3, 2 0)')
     = st_geomfromtext('LINESTRING(2 0, 2 3, 1 0)');
-- ==> true (was falsch ist - s. später)
-- aber:
select st_equals(
       st_geomfromtext('LINESTRING(1 0, 1 3, 2 0)'),
       st_geomfromtext('LINESTRING(2 0, 2 3, 1 0)')
);
-- ==> false
select st_disjoint(
       st_geomfromtext('LINESTRING(1 0, 1 3, 2 0)'),
       st_geomfromtext('LINESTRING(2 0, 2 3, 1 0)')
);
-- ==> false
select st_disjoint(
       st_geomfromtext('LINESTRING(1 0, 1 3, 2 0)'),
       st_geomfromtext('POINT(5 3)')
);
-- ==> true
```

5.2 Beispiel: Isolierte Objekte

Welche geometrischen Objekte überschneiden sich nicht mit welchen Flächen?

```
-- g2 ohne Einschränkung erzeugt in PostgreSQL 8.3 Speicherüberlauf
select g1.name, g2.name from mixed g1, mixed g2
  where st_disjoint(g1.geom, g2.geom)
  and st_geometrytype(g2.geom) ='ST_Polygon';
```

5.3 Schnitte und Überdeckungen

Funktion	Erklärung
`st_intersects(geometrie1, geometrie2)`	geometrie1 schneidet geometrie2
`st_crosses(geometrie1, geometrie2)`	geometrie1 schneidet geometrie2, die Dimension des Schnitts ist kleiner als die maximale Dimension der Geometrien
`st_overlaps(geometrie1, geometrie2)`	geometrie1 schneidet geometrie2, die Dimension des Schnitts ist gleich zur Dimension beider Geometrien
`st_contains(geometrie1, geometrie2)`	geometrie1 enthält echt geometrie2
`st_within(geometrie1, geometrie2)`	geometrie1 ist echt in geometrie2 enthalten
`st_touches(geometrie1, geometrie2)`	geometrie1 berührt geometrie2

5.4 Beispiel: Schnitt

Welche Linien schneiden A1?

```
1 select m1.name from mixed m1, mixed m2
2   where st_intersects(m1.geom, m2.geom)
3     and st_geometrytype(m1.geom)='ST_LineString'
4     and m2.name='A1';
```

5.5 Beispiele Überdeckung/Berührung

Welche Punkte sind in Fläche A3 enthalten?

```
1 select m1.name from mixed m1, mixed m2
2   where st_contains(m2.geom, m1.geom)
3     and st_geometrytype(m1.geom)='ST_Point'
4     and m2.name='A3';
5 -- oder
6 select m1.name from mixed m1, mixed m2
7   where st_within(m1.geom, m2.geom)
8     and st_geometrytype(m1.geom)='ST_Point'
9     and m2.name='A3';
```

Mit Einbeziehen der Randpunkte:

```
1 select m1.name from mixed m1, mixed m2
2   where
3     (st_within(m1.geom, m2.geom) or st_touches(m1.geom, m2.geom))
4     and st_geometrytype(m1.geom)='ST_Point'
5     and m2.name='A3';
```

Welche Flächen grenzen an A5?

```
select m1.name from mixed m1, mixed m2
  where st_touches(m1.geom, m2.geom)
    and st_geometrytype(m1.geom)='ST_Polygon'
    and m2.name='A5';
```

6 Indices und Bounding-Boxes

6.1 Geometrische Indices

- Test Gleichheit kompliziert
- Vereinfachung: Betrachten der *Bounding-Box*
- Ordnungsrelation kaum definierbar, damit schnelle Suche schwierig
- Abhilfe: spezielle Baumstrukturen

6.2 Boxen

PostGIS unterstützt einen eigenen Datentyp für achsenparallele, rechteckige Boxen: `box2d` (und analog `box3d`).

Anlegen einer Box-Spalte:

```
create table tableName (
  ...
  box box2d
  ...
);
```

Eingabe von Boxdaten:

```
insert into tableName (box)
  values ('BOX(x0 y0, x1 y1)');
```

Meist werden Boxen als Hilfskonstrukt gebraucht.

6.3 GIST-Index

Anlegen eines Index' über die Geometriespalte:

```
create index idxName on tabelle using gist(geometrieSpalte);
```

Jedes Geometrieobjekt erhält automatisch eine Bounding-Box:

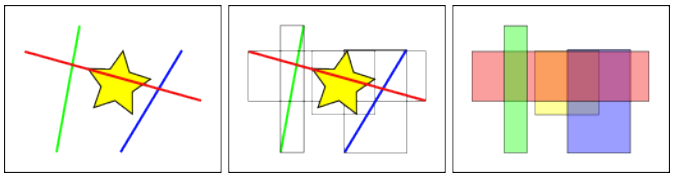

Abbildung 6.1: Konstruktion der Bounding-Box (Quelle: [4])

Die BoundingBoxen werden hierarchisch nach Gebieten zu größeren Boxen aggegiert. Suche muss nur noch in den jeweiligen BoundingBoxen erfolgen.

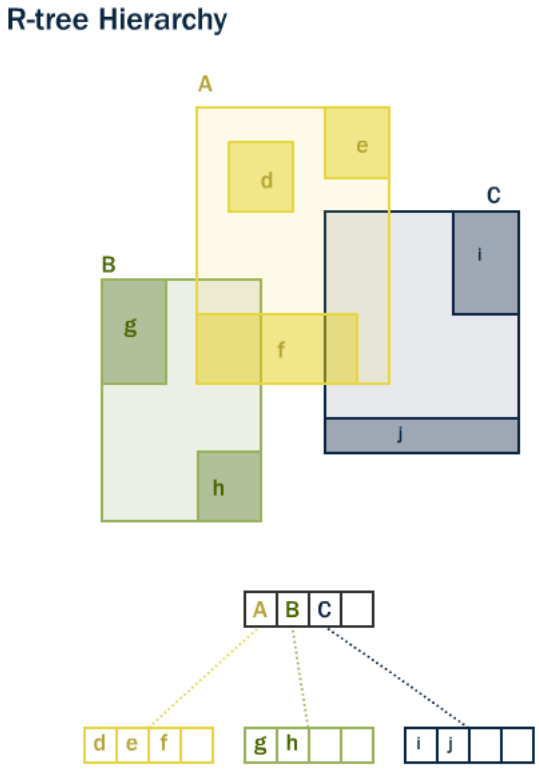

Abbildung 6.2: GiST-Index über Bounding-Boxen (Quelle: [4])

6.5 Erzeugen der Bounding-Box

Ermitteln der Bounding-Box für Geometrie-Objekte:

```
box2d(geometrieObjekt)
```

Beispiel:

```
select box2d(geom) from mixed;
```

Ergebnis:

```
     box2d
----------------
 BOX(2 2,6 6)
 BOX(1 6,10 10)
...
```

Erzeugen einer selbst definierten Bounding-Box:

```
st_makebox2d(linkerUntererPunkt, rechterObererPunkt)
-- oder:
'BOX(x0 y0, x1 y1)'::box2d
```

Beispiel:

```
select st_makebox2d(
  st_geomfromtext('POINT(0 0)',-1),
  st_geomfromtext('POINT(10 10)', -1)
);
-- alternativ:
select 'BOX(0 0, 10 10)'::box2d;
```

Ergebnis:

```
  st_makebox2d
----------------
 BOX(0 0,10 10)
```

Anmerkung 6.1. *BOX ist ein eigener Typ und kein Geometrietyp:*

```
select st_astext(
  st_makebox2d(
    st_geomfromtext('POINT(0 0)',-1),
    st_geomfromtext('POINT(10 10)', -1)
  )
);
-- alternativ:
select st_astext('BOX(0 0, 10 10)'::box2d);
```

Ergebnis:

```
             st_astext
-----------------------------------
 POLYGON((0 0,0 10,10 10,10 0,0 0))
```

6.6 Suche mit Bounding-Boxen

Die meisten Geometrie-Operatoren benutzen Bounding-Boxen als Filter.

Spezielle Bounding-Box-Operatoren sind u.a.

Operator	**Bedeutung**
`geomObjekt1 = geomObjekt2`	Bounding-Boxen sind identisch
`geomObjekt1 && geomObjekt2`	Bounding-Boxen überschneiden sich
`geomObjekt1 @ geomObjekt2`	Bounding-Box von Objekt 1 ist in Box von Objekt 2 enthalten

Es existiert eine große Zahl weiterer Operatoren für Bounding-Boxen.

Beispiel: Welche Linienzüge schneiden welche Polygone (Bounding-Box-Test)?

```
select a.name,b.name from mixed a, mixed b
  where a.geom && b.geom
    and st_geometrytype(a.geom)='ST_LineString'
    and st_geometrytype(b.geom)='ST_Polygon';
```

Ergebnis:

```
 name | name
------+------
 L1   | A1
 L1   | A2
 L1   | A3
 L1   | A4
 L2   | A1
 L2   | A2
 L2   | A3
```

7 Referenzsysteme, Transformationen und Abstände

7.1 Projektionen

- Bisher sind wir von karthesischen Koordinaten in der Euklidschen Geometrie ausgegangen.
- Geodaten sind aber auf der Erde angesiedelt und müssen referenziert werden.
- Angenähertes Bezugssystem auf der Erde über Näherungskugel oder -ellipsoid (WGS84)
- Projektion auf Euklidsche Ebene:
  - Winkeltreu (Elliptical, Spherical Mercator)
  - Abstandstreu (Gauß-Krüger, UTM)
  - ...
- Sammlung aller benutzten Projektionen und ihrer Parameter durch die EPSG
- Umprojektionen nicht trivial
- Werkzeug: *Proj4*

Anmerkung 7.1. *Abstands- und winkeltreue Abbildungen der Kugeloberfläche in die Ebene sind nur für kleine Gebiete und nur näherungsweise möglich.*

7.2 Wichtige EPSG-Codes

Code	Bezugssystem bzw. Projektion
`4326`	WGS84, weltweiter Ellipsoid, Angabe in Breiten-/Längengraden
`900913, 3857`	Google Spherical Mercator Projection Karthesisches Koordinatensystem in (verzerrten) Metern näherungsweise winkeltreu
`31468`	Gauß-Krüger, Zone 4 (12.-15. Längengrad) Karthesisches Koordinatensystem, auf 3-Grad-Streifen beschränkt, Bezugssystem: Bessel-Ellipsoid mit Datum Potsdam abstands- und winkeltreu
`25833`	UTM, Zone 33N (12.-18. Längengrad) Karthesisches Koordinatensystem, auf 6-Grad-Streifen beschränkt WGS84-Ellipsoid, abstands- und winkeltreu

OSM benutzt wie *Google Maps* die *Spherical Mercator Projection* für die sogenannten *Slippy Maps*. Aus diesem Grund sind die anzeigten Entfernungsskalen und Maßstabsangaben meist falsch.

7.3 Umprojektionen

Warnung 7.2. Derzeit ist *Proj4* in *Debian Lenny* defekt (die folgenden Operationen zeigen z.T. unsinnige Ergebnisse). Statt dessen muss das Paket aus den *Backports* installiert werden.

```
st_transform(geometrieObjekt, epsgCode)
```

Beispiel: Chemnitz (WGS84: 12.917E, 50.833N) in Google- -> UTM- -> WGS-84-Koordinaten.

```
-- WGS84 => Google
select st_astext(
  st_transform(
    st_geomfromtext('POINT(12.917 50.833)', 4326),
    900913
  )
);
-- => POINT(1437913.86257672 6591806.3505577)
-- Google => UTM
select st_astext(
  st_transform(
    st_geomfromtext('POINT(1437913.86257672 6591806.3505577)', 900913),
    25833
  )
);
-- => POINT(353319.166645782 5633321.74489742)
-- UTM => WGS84
select st_astext(
  st_transform(
    st_geomfromtext('POINT(353319.166645782 5633321.74489742)', 25833),
    4326
  )
);
-- => POINT(12.9169999999973 50.8329999999921)
```

7.4 Abstände

`st_distance()` funktioniert nicht unbedingt korrekt:
Beispiel: Entfernung Chemnitz (WGS84: 12.917E, 50.833N) - Dresden (WGS84: 13.738E, 51.049N)

```
select st_distance(
  st_geomfromtext('POINT(12.917 50.833)',4326),
  st_geomfromtext('POINT(13.738 51.049)',4326)
);
-- => 0.848938749262866
```

Abhilfe:

- Entfernungsfunktionen für sphärische Koordinaten:

  ```
  st_distance_sphere(punkt1, punkt2)
  ```

  Bis PostGIS 1.5 nur für Punkte implementiert.
- Abstandstreue Umprojektion in karthesische Ebene

Beispiel: Abstand Chemnitz-Dresden (Luftlinie)

```
select st_distance_sphere(
  st_geomfromtext('POINT(12.917 50.833)',4326),
```

```
  st_geomfromtext('POINT(13.738 51.049)',4326)
);
-- => 62336.5594297526
-- Umprojektion nach UTM
select st_distance(
  st_transform(
    st_geomfromtext('POINT(12.917 50.833)',4326),
    25833
  ),
  st_transform(
    st_geomfromtext('POINT(13.738 51.049)',4326),
    25833
  )
);
-- => 62493.9257541902
-- Umprojektion nach Gauß-Krüger
select st_distance(
  st_transform(
    st_geomfromtext('POINT(12.917 50.833)',4326),
    31468
  ),
  st_transform(
    st_geomfromtext('POINT(13.738 51.049)',4326),
    31468
  )
);
-- => 62514.9570308865
-- Umprojektion nach Google
select st_distance(
  st_transform(
    st_geomfromtext('POINT(12.917 50.833)',4326),
    900913
  ),
  st_transform(
    st_geomfromtext('POINT(13.738 51.049)',4326),
    900913
  )
);
-- => 99039.8001451806 (Unsinn)
```

8 Aggregatfunktionen

8.1 Ziel

SQL kennt Funktionen zum Zusammenfassen von Daten:

- SUM()
- COUNT()
- AVG()
- ...

Für Geometrien weitere Zusammenfassungen sinnvoll:

- Zusammenfassen von Einzelgeometrien zu neuer Geometrie

- Linie aus Einzelpunkten
- Bounding-Box über ausgewählte Elemente
- ...

Bsp: OSM-Schema verwaltet Stützstellen eines Linienzugs in eigener Tabelle. Schneidet der Linienzug ein bestimmtes Gebiet?

Häufig sinnvoll mit `GROUP BY`-Klauseln kombinierbar.

8.2 Bounding-Box

In welche Bounding-Box fallen alle gewählten Elemente?

```
st_extent(geometriespalte)
```

Beispiel:

```
-- Bounding-Box aller Elemente:
select st_extent(geom) from mixed;
-- => BOX(0 0,10 11)
-- Bounding-Box aller Punkte:
select st_extent(geom) from mixed where st_geometrytype(geom)='ST_Point';
-- => BOX(1 4,8 9)
```

8.3 Zusammenfassen von Geometrien

Notwendig, um Funktionen für ein Geometrieobjekt auf eine Gruppe anwenden zu können.

```
st_collect(geometriespalte)
```

Ergebnis:

- `MULTI`-Objekt bei gleichartigen Geometrieobjekten
- `GEOMETRYCOLLECTION` bei verschiedenartigen Geometrieobjekten

```
-- Verschiedene Objekte:
select st_astext(st_collect(geom)) from mixed;

-- Ergebnis:
GEOMETRYCOLLECTION(
  POLYGON((4 2,6 6,2 6,4 2)),
  POLYGON((2 6,6 6,10 10,1 10,2 6)),
  POLYGON((4 2,8 2,10 10,6 6,4 2)),
  POLYGON((0 6,3 0,4 2,2 6,1 10,0 6)),
  POLYGON((3 0,8 2,4 2,3 0)),
  POINT(6 9),POINT(1 6),POINT(8 7),POINT(5 4),POINT(7 4),
  LINESTRING(2 11,5 8,4 5,4 4,7 3),
  LINESTRING(6 8,8 5)
)
```

```
-- Gleiche Objekte:
select st_astext(st_collect(geom)) from mixed
  where st_geometrytype(geom)='ST_Point';

-- Ergebnis:
MULTIPOINT(6 9,1 6,8 7,5 4,7 4)
```

Verwendung z.B. für:

Funktion	**Bedeutung**
`ST_ConvexHull(geometrieobjekt)`	Konvexe Hülle (Polygon)
`ST_Centroid(geometrieobjekt)`	Schwerpunkt
`ST_Buffer(geometrieobjekt, abstand)`	Polygon mit gegebenen Abstand zur Geometrie (s. vorn)

Beispiel: Schwerpunkt der Geometrieobjekte.

```
select st_astext(st_centroid(st_collect(geom))) from mixed;
-- Ergebnis:
POINT(4.83333333333333 5.83333333333333)
```

Beispiel: Puffer um einen Punkt bzw. gesamte Geometrie.

```
select st_astext(st_buffer(geom, 2)) from mixed
  where name='P1';
select st_astext(st_buffer(st_collect(geom), 2)) from mixed;
```

8.4 Vereinigungen

```
st_union(geometriespalte)
```

Ergebnis ist eine Geometrie abhängig von den Eingangsdaten: `MULTI`-Objekt, `GEOMETRYCOLLECTION`, `POLYGON`.

Beispiele:

```
-- Mixed:
select st_astext(st_union(geom)) from mixed;
-- Ergebnis:
GEOMETRYCOLLECTION(
  LINESTRING(2 11,3 10),
  POLYGON((8 2,3 0,0 6,1 10,3 10,10 10,8 2))
)
-- Polygone:
select st_astext(st_union(geom)) from mixed
  where st_geometrytype(geom) = 'ST_Polygon';
-- Ergebnis:
POLYGON((8 2,3 0,0 6,1 10,10 10,8 2))
```

8.5 Linienzüge

Punktmengen können in Linienzüge umgewandelt werden.

```
ST_MakeLine(geometriespalte)
```

Beispiel:

```
select st_astext(st_makeline(geom)) from mixed
  where st_geometrytype(geom) = 'ST_Point';
-- Ergebnis:
LINESTRING(6 9,1 6,8 7,5 4,7 4)
```

Umformung in Polygon ist möglich, der Linestring muss dazu noch geschlossen werden:

```
st_addpoint(line, st_startpoint(line))
```

Beispiel:

```
select st_astext(
  st_makepolygon(
    st_addpoint(sub.line, st_startpoint(sub.line))
  )) from (
    select st_makeline(geom) as line from mixed
      where st_geometrytype(geom) = 'ST_Point'
  ) as sub;
-- Ergebnis:
POLYGON((6 9,1 6,8 7,5 4,7 4,6 9))
```

9 OpenStreetMap und PostGIS

9.1 Übersicht

OpenStreetMap verwendet (mindestens) zwei unterschiedliche Datenbank-Schemata:

OSM-DB Verwaltung der Rohdaten (Punkte, Wege, Relationen, ...), Zugang über XML-API, vorrangig zum Editieren der Daten

Mapnik-DB Aufbereitete Daten zum Rendern von Karten/Kacheln mit Mapnik

9.2 Datenimport

- Bereitstellung im XML-Format
- Kleine Datenmengen können direkt über die API bezogen werden: `http://api.openstreetmap.org/`
- Große Datenmengen werden wöchentlich aktuell im Planetfile bereitgestellt: `http://planet.openstreetmap.org/`, dazu tägliche, stündliche und minütliche Diffs.
- Daten Europa (gesamt und Einzelstaaten) täglich aktuell über `http://download.geofabrik.de/`

Neu: Downloads im ProtocolBufBinary-Format (nur Europa) `http://wiki.openstreetmap.org/wiki/ProtocolBufBinary` (wesentlich kleiner, schneller auszuwerten.

9.3 Osmosis

- Werkzeug zum Konvertieren von Datenformaten und Zuschneiden des benötigten Gebietes.
- Java-1.6-Anwendung
- Kann Datenbankschema im OSM-Format erzeugen und Daten importieren sowie exportieren
- Download: `http://dev.openstreetmap.org/~bretth/osmosis-build/osmosis-latest.tgz`

Ausschneiden:

```
osmosis --rx file="OSM-Datei" \
  --bb left=lon0 bottom=lat0 right=lon1 top=lat1 \
  clipIncompleteEntities=true \
  --wx file="ausschnittsdatei"
```

Anmerkung 9.1. *Benötigtes Gebiet so klein wie möglich wählen, da die Datenmengen extrem groß sind.*

Ausschnitt eines 1x1°-Ausschnitts aus Europa dauerte ca. 40 Minuten (12 GB RAM).

Schema erzeugen:

```
psql -h localhost -U owner dbname -f osmosisPfad/script/pgsql_simple_schema_0.6.sql
```

Daten in OSM-DB schreiben:

```
osmosisPfad/bin/osmosis --rx file="OSM-Datei" \
  --wp host="localhost" database="dbname" user="user" password="password"
```

9.4 OSM-Schema

Geographisches Bezugssystem: 4326 (WGS84)

```
 Schema |       Name       |   Typ   | Eigentümer
--------+------------------+---------+------------
 public | nodes            | Tabelle | osm
 public | relation_members | Tabelle | osm
 public | relations        | Tabelle | osm
 public | schema_info      | Tabelle | osm
 public | users            | Tabelle | osm
 public | way_nodes        | Tabelle | osm -- Verknüpfungstabelle Way-Nodes
 public | ways             | Tabelle | osm
```

Aufbau `ways`:

```
    Spalte    |             Typ             | Attribute
--------------+-----------------------------+-----------
 id           | bigint                      | not null
 version      | integer                     | not null
 user_id      | integer                     | not null
 tstamp       | timestamp without time zone | not null
 changeset_id | bigint                      | not null
 tags         | hstore                      |
 nodes        | bigint[]                    |
Indexe:
    »pk_ways« PRIMARY KEY, btree (id)
```

OSM kennt keine Polygone als Geometrietyp - wird über Tags entschieden, falls der Linienzug geschlossen ist.

9.5 Beispiel

Kneipen in der Nähe des Hexenbergs Grüna:

```
select id, tags->'name' as name, st_asewkt(geom)
  from nodes
    where tags ? 'amenity' and tags->'amenity'='restaurant'
    and st_distance_sphere(
      geom,
      st_geomfromtext('POINT(12.7974 50.8170)',4326)
    ) < 1000;
```

Ergebnis:

```
    id     |      name       |               st_asewkt
-----------+-----------------+----------------------------------------
  78609332 | Forsthaus Grüna | SRID=4326;POINT(12.7986111 50.8213083)
 376329307 | Grünaer Hof     | SRID=4326;POINT(12.7877495 50.8115585)
 376329318 | Folklorehof     | SRID=4326;POINT(12.7920205 50.8148084)
```

Tags werden als *HStore* (assoziatives Array bzw. Hash) verwaltet. Zugriff:

Operator	Bedeutung
`spalte?'key'`	prüft, ob Schlüssel `key` vorhanden ist
`spalte->'key'`	Liefert den Wert zum Schlüssel `key`

9.6 Beispiel: Knoten eines Wegs

```
select n.tags->'name' as name, st_astext(n.geom)
  from nodes n
    join ways as w
      on n.id = any(w.nodes) -- Array w.nodes enthält Node-IDs
  where  w.tags->'name'='Reichenhainer Straße';
```

Ergebnis:

```
                    name                    |          st_astext
--------------------------------------------+-----------------------------
                                            | POINT(12.9321111 50.8126274)
                                            | POINT(12.9365026 50.8076915)
 Städtischer Friedhof                       | POINT(12.9342639 50.810661)
                                            | POINT(12.9363514 50.80792)
...
```

Alternativ über Verknüpfungstabelle `way_nodes`.

```
select n.tags->'name' as name, st_astext(n.geom)
   from nodes n
     join way_nodes wn on wn.node_id=n.id
     join ways w on wn.way_id=w.id
   where
     w.tags->'name'='Reichenhainer Straße'
   order by wn.sequence_id;
```

9.7 Osm2pgsql und Mapnik

Tool zum Importieren von OSM-Daten in eine PostGIS-Datenbank mit *Mapnik*-Schema zum Rendern von Karten.

Ausführliche Anleitung (deutsch):
`http://wiki.openstreetmap.org/wiki/DE:HowTo_minutely_hstore`
`http://wiki.openstreetmap.org/wiki/DE:HowTo_Mapnik_%26_Tirex`

Installation:

```
sudo apt-get build-dep osm2pgsql
svn co http://svn.openstreetmap.org/applications/utils/export/osm2pgsql/
```

```
cd osm2pgsql
./autogen.sh
./configure
make
make install
```

Datenbank benötigt Spherical-Mercator-Projektionsdaten:

```
su - postgres psql dbname -f 900913.sql
```

Datenimport:

```
osm2pgsql --create # Datenbank erzeugen \
   --database dbname --host localhost --username dbuser --password # Verbindungsparameter \
  --prefix planet # Tabellenpräfix \
  --cache 2048 # Cache size in MB, für Planet >= 8GB wählen \
  --hstore # hstore verwenden \
  OSM-Datei
```

Anmerkung 9.2. *Slim-Mode erlaubt regelmäßiges Datenbank-Update, ist aber sehr langsam. Kompletter Neuimport ist schneller.*

9.8 Mapnik-Schema

Datenbank für das Rendern optimiert:

- Geographisches Bezugssystem: 900913 (Spherical Mercator Projection)
- Referenzen zu Koordinaten aufgelöst
- Trennung von Linien und Polygonen
- Derzeit nicht benutzte Tags in HStore verbannt
- Abtrennen von speziellen Linien (Roads) für niedrige Zoomstufen
- Schnelleres Rendern
- Beziehung Way-Nodes geht verloren
- Keine Routing-Berechnung möglich
- Keine (direkte) Abstandsbestimmung möglich

Schema:

```
 Schema |       Name       |   Typ   | Eigentümer
--------+------------------+---------+------------
 public | planet_line      | Tabelle | mapnik
 public | planet_point     | Tabelle | mapnik
 public | planet_polygon   | Tabelle | mapnik
 public | planet_roads     | Tabelle | mapnik
```

Aufbau `planet_line`:

```
       Spalte       |   Typ    | Attribute
--------------------+----------+-----------
 osm_id             | integer  |
```

```
 access             | text     |
 addr:flats         | text     |
 addr:housenumber   | text     |
 addr:interpolation | text     |
 admin_level        | text     |
 aerialway          | text     |
 aeroway            | text     |
 amenity            | text     |
...
 waterway           | text     |
 width              | text     |
 wood               | text     |
 z_order            | integer  |
 way_area           | real     |
 tags               | hstore   |
 way                | geometry |
Indexe:
    »planet_line_index« gist (way)
```

9.9 Mapnik

- Python-Bibliothek
- Renderer für Karten
- Datenquellen u.a. PostGIS und Shapefiles
- Über XML-Dateien konfigurierbar
- Ausgabeformate: PNG, JPEG, SVG, PostScript, PDF, ...
- Sinnvollerweise selbst übersetzen

Installation und Konfiguration: `http://wiki.openstreetmap.org/wiki/DE:HowTo_Mapnik_%26_Tirex`

Konfiguration über Template-Dateien im `inc`-Verzeichnis (Datenbankparameter, Fonts, Verzeichnisse, Projektion)

Küstenlinien und globale Grenzen beschaffen.

```
./get-coastlines.sh
```

Vereinfachung des Aufrufs bzw. weiteres Beispielscript:

```
wget http://svn.toolserver.org/svnroot/mazder/mapnik-in-a-box/tools/osm-render
```

9.10 Mapnik-Stylefile

Aufbau:

- Style beschreibt das Rendern
- Filter und Zoomlevel erlauben gezielte Auswahl
- Layer beschreibt die Datenquelle

- SQL-Abfrage für benötigte Daten

Beispiel: Anzeige aller Straßen und Autobahnen.

samples/highways.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Map [
<!-- Pfad anpassen! -->
<!ENTITY % entities SYSTEM "/data/osm/osm-mapnik/inc/entities.xml.inc">
%entities;
]>
<!-- This stylesheet uses features only available in mapnik builds with
     libxml2 as the XML parser. Furthermore, there are other features
     and behaviour that necessitate an upgrade to mapnik 0.7.1 -->
<Map bgcolor="transparent" srs="&srs900913;" minimum_version="0.7.1">
  &fontset-settings;
<Style name="roads">
    <Rule>
      <Filter>[highway] = 'motorway' or [highway] = 'motorway_link'</Filter>
      &maxscale_zoom8;
      &minscale_zoom18;
      <LineSymbolizer>
        <CssParameter name="stroke">red</CssParameter>
        <CssParameter name="stroke-width">2</CssParameter>
      </LineSymbolizer>
    </Rule>
    <Rule>
      <Filter>[highway] != 'motorway' and [highway] != 'motorway_link'</Filter>
      &maxscale_zoom8;
      &minscale_zoom18;
      <LineSymbolizer>
        <CssParameter name="stroke">orange</CssParameter>
        <CssParameter name="stroke-width">1</CssParameter>
      </LineSymbolizer>
    </Rule>
    <Rule>
      <Filter>[railway] !=''</Filter>
      &maxscale_zoom8;
      &minscale_zoom18;
      <LineSymbolizer>
        <CssParameter name="stroke">black</CssParameter>
        <CssParameter name="stroke-width">1.5</CssParameter>
      </LineSymbolizer>
    </Rule>
</Style>
<Layer name="roads" status="on" srs="&osm2pgsql_projection;">
    <!--StyleName>roads-casing</StyleName-->
    <StyleName>roads</StyleName>
    <Datasource>
      <Parameter name="table">
      (select way,highway,railway,
       case when tunnel in ('yes','true','1') then 'yes'::text else tunnel end as tunnel
       from &prefix;_roads
       where highway is not null
          or (railway is not null and (service is null or service not in ('spur','siding','yard')))
       order by z_order
      ) as roads
      </Parameter>
      &datasource-settings;
    </Datasource>
```

```
56 </Layer>
57
58 </Map>
```

Rendern:

```
1 ./osm-render -f highways -b 12,50,13,51 -x400x600 -s highways.xml
```

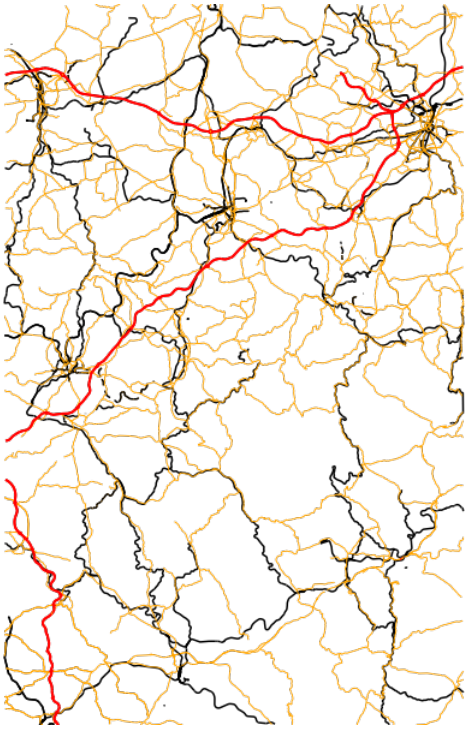

Abbildung 9.1: Ergebnis (Bounding-Box: 12, 50, 13, 51)

10 Schnittstellen zu Programmiersprachen

10.1 Allgemeines

Prinzipiell kann normale PostgreSQL-API der jeweiligen Programmiersprache verwendet werden:

- Elemente der Geometriespalte können mit Hilfe der PostGIS-Konvertierungsfunktionen in Strings umgewandelt und als solche gelesen oder geschrieben werden.
- Alternativ: Binärformat verwenden (wer es versteht)
- Einige Programmiersprachen bieten Bibliotheken zur automatischen Konvertierung in adequate Datenstrukturen

10.2 Java-API

Benötigte Bibliotheken - Bezug von `http://www.postgres.org` - `http://www.postgis.org`

- `postgres*.jar` - JDBC-Treiber
- `postgis*.jar` - PostGIS-Wrapperobjekte

Geometrieobjekte können direkt abgefragt und in den Typ `PGgeometry` umgecastet werden:

```
...
PreparedStatement stm = conn.prepareStatement("select ...");
...
ResultSet rs = rs.executeQuery();
while(rs.next()) {
  PGgeometry pggeom = (PGgeometry)rs.getObject(index);
}
```

Dieses Wrapperobjekt enthält das eigentliche Geometrie-Objekt, dass in den passenden Typ umgecastet werden kann:

```
Geometry geom = pggeom.getGeometry();
if (geom instanceof Point) {
  Point p = (Point)geom;
  // Verabeite p
  ...
}
else if (...) {
  ...
}
```

10.3 Beispiel

Anzeige aller Geometrieobjekte der Tabelle `mixed`.

`samples/postgis1/src/Postgis1.java`

```
public class Postgis1 {
  JFrame frame;
  GeomPanel panel;
  ArrayList<Shape> shapes = new ArrayList<Shape>();
  final static int SCALE = 30;

  // Panel for drawing geometry objects
  class GeomPanel extends JPanel {
    public GeomPanel() {
      super();
      setPreferredSize(new Dimension(400, 400));
    }

    @Override
    protected void paintComponent(Graphics g) {
```

```
40        super.paintComponent(g);
41        Graphics2D g2 = (Graphics2D) g;
42        // Flip on horizontal axis
43        g2.scale(1, -1);
44        // Move down (10 points border)
45        g2.translate(10, -390);
46        for (Shape shape : shapes) {
47          if (shape == null)
48            continue;
49          g2.draw(shape);
50        }
51      }
52    }
53
54    public Postgis1() {
55      JFrame.setDefaultLookAndFeelDecorated(true);
56      frame = new JFrame("PostGIS Test");
57      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
58      panel = new GeomPanel();
59      frame.add(panel, BorderLayout.CENTER);
60      frame.pack();
61      frame.setVisible(true);
62    }
63
64    public Shape polygonShape(Polygon pl) {
65      System.out.println("Polygon " + pl);
66      // Works only if the first ring is the outer one!
67      // Alternatively draw all rings from 0 to pl.numRings()-1
68      LinearRing l = pl.getRing(0);
69      Path2D path = new Path2D.Double();
70      Point p = l.getFirstPoint();
71      path.moveTo(SCALE * p.x, SCALE * p.y);
72      for (int i = 1; i < l.numPoints(); i++) {
73        p = l.getPoint(i);
74        System.out.println(p);
75        path.lineTo(SCALE * p.x, SCALE * p.y);
76      }
77      return path;
78    }
79
80    public Shape lineShape(LineString l) {
81      System.out.println("LineString " + l);
82      Path2D path = new Path2D.Double();
83      Point p = l.getFirstPoint();
84      path.moveTo(SCALE * p.x, SCALE * p.y);
85      for (int i = 1; i < l.numPoints(); i++) {
86        p = l.getPoint(i);
87        System.out.println(p);
88        path.lineTo(SCALE * p.x, SCALE * p.y);
89      }
90      return path;
91    }
92
93    public Shape pointShape(Point p) {
94      System.out.println("Point " + p);
95      double x = SCALE * p.x;
96      double y = SCALE * p.y;
97      return new Ellipse2D.Double(x - 3, y - 3, 6, 6);
98    }
99
100   public void go() {
```

```
    Connection conn;
    try {
      Class.forName("org.postgresql.Driver");
      String url = "jdbc:postgresql://localhost:5432/postgis";
      conn = DriverManager.getConnection(url, "postgis", "postgis-pw");
      PreparedStatement stm = conn
          .prepareStatement("select geom from mixed order by name");
      ResultSet rs = stm.executeQuery();
      while (rs.next()) {
        PGgeometry pgeom = (PGgeometry) rs.getObject(1);
        Geometry geom = pgeom.getGeometry();
        if (geom instanceof Polygon)
          shapes.add(polygonShape((Polygon) geom));
        else if (geom instanceof LineString)
          shapes.add(lineShape((LineString) geom));
        else if (geom instanceof Point)
          shapes.add(pointShape((Point) geom));
      }
      rs.close();
      stm.close();
      panel.repaint();
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }

  public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
      @Override
      public void run() {
        new Postgis1().go();
      }
    });
  }
}
```

Ergebnis:

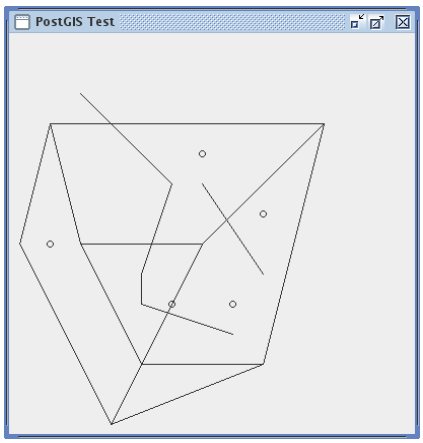

Abbildung 10.1: Screenshot

11 Literatur

[1] `http://postgis.refractions.net/` - PostGIS Home

[2] `http://postgis.refractions.net/docs/` - PostGIS Dokumentation

[3] `http://www.opengeospatial.org/standards/sfs` - Spezifikation OpenGIS (Simple Features - SQL Option)

[4] `http://workshops.opengeo.org/postgis-intro/` - PostGIS-Tutorial

[5] `http://www.postgisonline.org/tutorials/` - Tutorials

Abkürzungen

EPSG *European Petroleum Survey Group Geodesy*
jetzt *Surveying and Positioning Committee* der *International Association of Oil & Gas Producers*

GIS *Geographisches Informations-System*
Programm zum Bearbeiten, Anzeigen und Konvertieren geographischer Informationen, Erzeugen von Karten etc.

GiST *Generalized Search Tree*
Datenbank-Index, Verallgemeinerung von B+-Bäumen.

OGC *Open Geospatial Consortium, Inc.*

OSM *OpenStreetMap*

SRID *Spatial Reference system IDentifier*

SRS *Spatial Reference System*
Geographisches Bezugssystem, sphärisch auf Ellipsoid oder karthesisch in einer Projektion.

WKB *Well Known Binary*
OGC-Spezifikation für die Binärdarstellung geographischer Daten.

WKT *Well Known Text*
OGC-Spezifikation für die Textdarstellung geographischer Daten.

WMS *Web Map Service*
Webservice zum Bereitstellen von Kartendaten.