



Stadt Uster, Abteilung Bau
GIS Kompetenzzentrum

Einführung PostGIS

PostGIS-Einführung Architektur und Funktionen

Andreas Neumann
GIS Kompetenzzentrum

Stadt Uster, Abteilung Bau
andreas.neumann@stadt-uster.ch, www.uster.ch

enthält Folien von Refrations Research
und Grafiken von Christian Strobl

Was ist PostGIS?

**Räumliche Erweiterung von PostgreSQL,
entwickelt von Refractions Research
(Victoria, BC, Canada)**

- ◆ **Open Source: GPL Lizenz**
- ◆ **Aktive Mailingliste und User Community**
- ◆ **Neue räumliche Datentypen**
- ◆ **Räumlicher Index (r-tree, bounding boxen)**
- ◆ **Räumliche Operatoren**
- ◆ **ca. 300 räumliche Funktionen**
- ◆ **Projektions-Support über Proj4**
- ◆ **Erweiterbar**

Kleine PostGIS Geschichte

7 Jahre Entwicklungsgeschichte

- ◆ 2001: Erster Release, Mapserver support
- ◆ 2002: Verbesserte Funktionen, R-Tree Indizes
- ◆ 2003: GEOS support, viele neue Funktionen
- ◆ 2004: Simple Features for SQL conformance
- ◆ 2005: Lightweight geometries
- ◆ 2006: OpenGIS SFSQL compliance
- ◆ 2007: SQL/MM, curves & performance

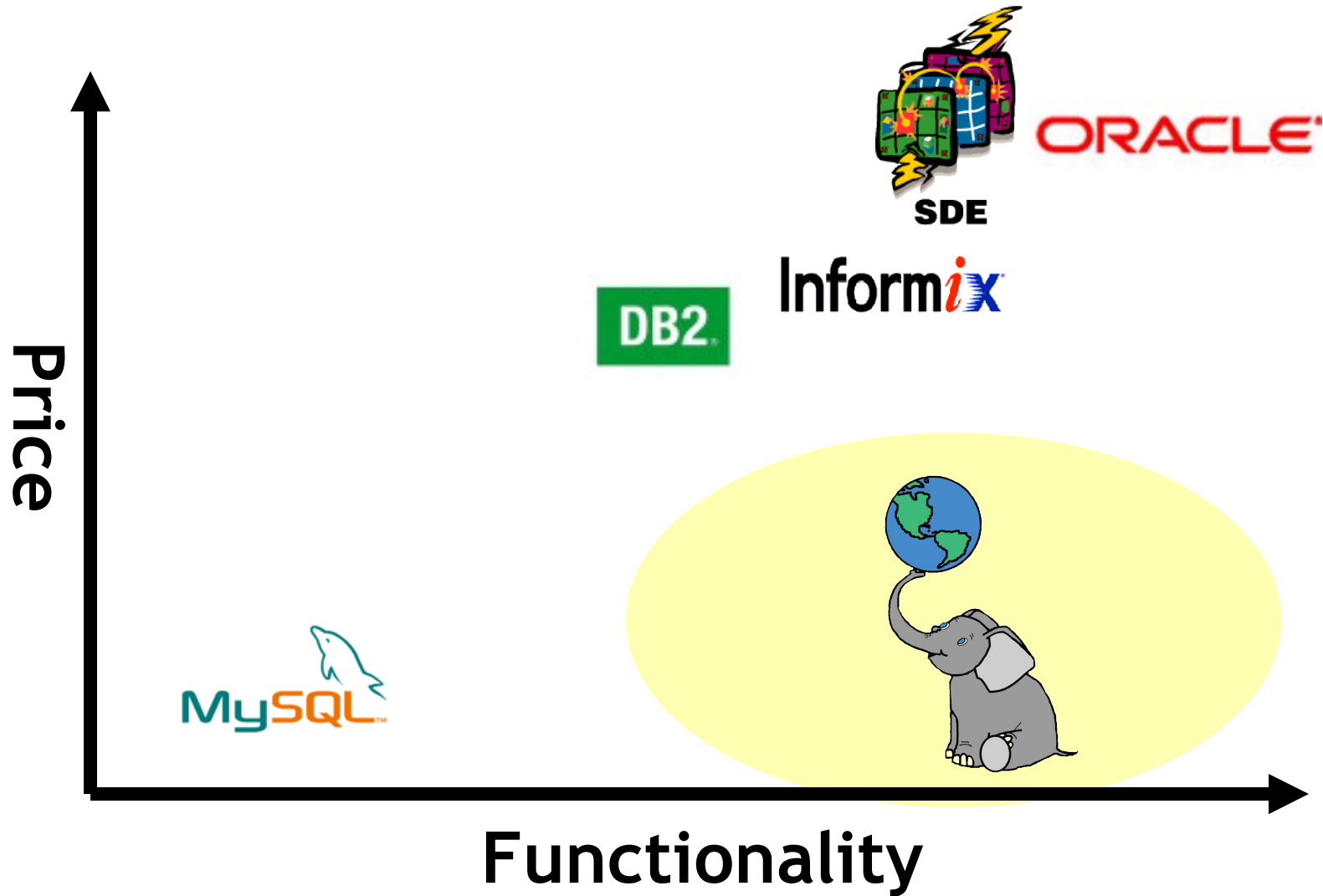
PostGIS Clients

- UMN Mapserver
- Geotools (Geoserver, uDig)
- QGIS, GRASS
- FDO (Mapguide, Autodesk Map 3D)
- JUMP (OpenJUMP, Kosmo)
- OGR
- FME (ArcGIS Data Interoperability Extension)
- Cadcorp SIS
- Manifold
- ESRI ArcSDE 9.3
- NASA Worldwind, Google Earth, MS Virtual Earth
- Python / Perl / PHP

Einige PostGIS Benutzer

- IGN Frankreich
- United States Postal Service
- Library of Congress
- DLR, NASA
- National Institute for Space Research, Brazil
- European Commission (Joint Research Center)
- States of BC, Yukon Territory, Saskatchewan, North Dakota, Colorado, Minnesota, Massachusetts, Canadian Forest Service
- Verkehrsministerium Holland
- Electricité de France
- Swisstopo/KOGIS
- Kantone Solothurn, Graubünden, Thurgau

Warum PostGIS?



Warum PostGIS?

Scalability*

“Enterprise”	1 Dual-Core	2 Quad-Core
Oracle	\$40,000	\$160,000
IBM DB2	\$36,400	\$145,600
MS SQL Server	\$25,000	\$50,000
IBM Informix	\$50,000	\$200,000
PostgreSQL	\$0	\$0

Preise variieren; Preise für „Spatial-Variante“, nicht OEM!

Warum PostGIS?

Keep it Simple ...

- ◆ **PostGIS Polygon**
 - **POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))**
- ◆ **Oracle Polygon**
 - **MDSYS.SDO_GEOMETRY(
2003, NULL, NULL,
MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1),
MDSYS.SDO_ORDINATE_ARRAY(0,0, 0,1, 1,1, 1,0,
0,0))**

PostGIS Architektur

Verschiedene Clients:
GIS Clients, DB-Admin Clients, Web-Clients, Eigene Programme

Postgis Data Types

Postgis Functions a. Operators

Spatial Ref Sys

PL/PGSQL (Procedural Language)

PostgreSQL Database Management System

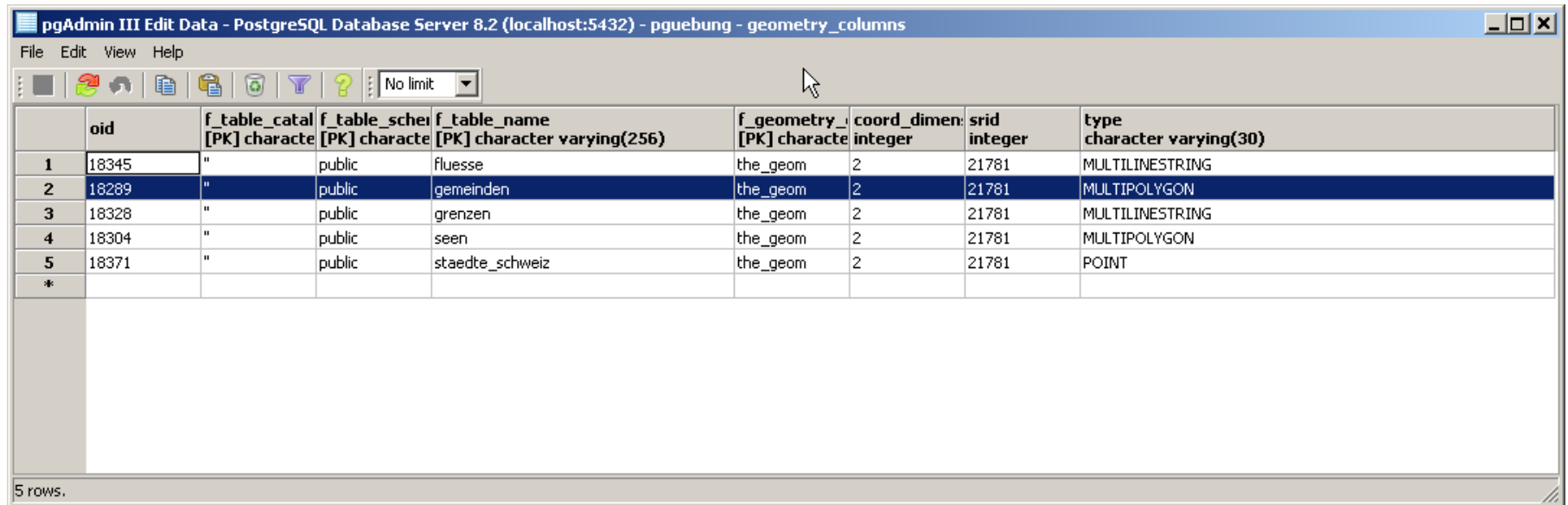
GEOS Geometry Library (C++)

Proj4 Projection Library (C)

Utilities:
Shp2pgsql,
postgis_restore.pl

PostGIS Systemtabellen

geometry_columns – list of spatial columns, SRIDs, and geometry data types in the database



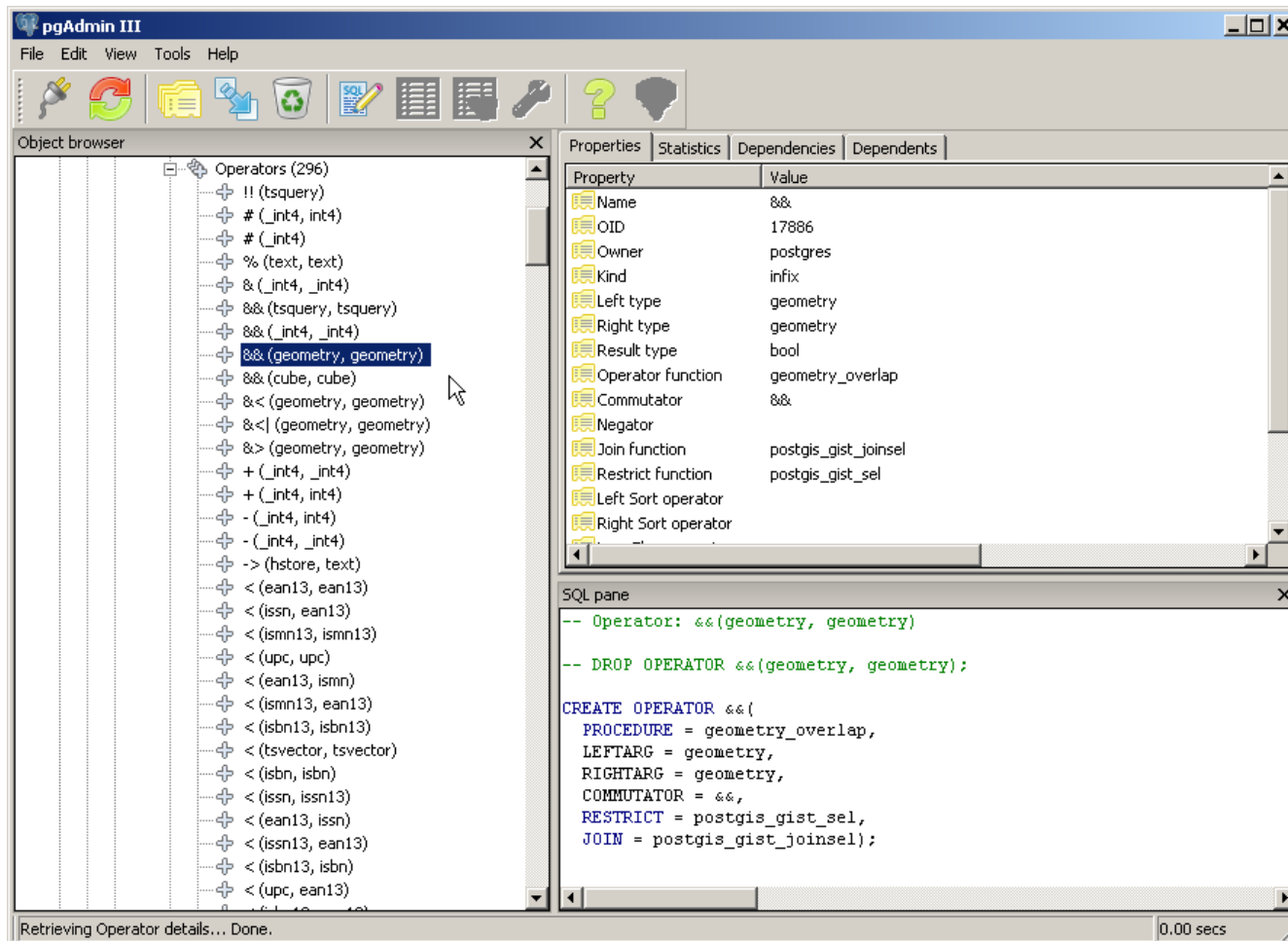
The screenshot shows the pgAdmin III interface with the 'geometry_columns' table selected. The table contains 5 rows of data, including columns for oid, f_table_catalog, f_table_schema, f_table_name, f_geometry_column, coord_dimension, srid, and type. The data is as follows:

	oid	f_table_catalog [PK] character	f_table_schema [PK] character	f_table_name [PK] character varying(256)	f_geometry_column [PK] character	coord_dimension integer	srid integer	type character varying(30)
1	18345	"	public	fluesse	the_geom	2	21781	MULTILINESTRING
2	18289	"	public	gemeinden	the_geom	2	21781	MULTIPOLYGON
3	18328	"	public	grenzen	the_geom	2	21781	MULTILINESTRING
4	18304	"	public	seen	the_geom	2	21781	MULTIPOLYGON
5	18371	"	public	staedte_schweiz	the_geom	2	21781	POINT
*								

5 rows.

PostGIS Räumliche Operatoren

Ansicht in pgAdmin III



The screenshot shows the pgAdmin III interface. On the left, the 'Object browser' pane displays a list of operators. The operator '<&& (geometry, geometry)' is selected. On the right, the 'Properties' pane shows the details of this operator. Below the properties, the 'SQL pane' displays the SQL code for creating and dropping this operator.

Property	Value
Name	<&&
OID	17886
Owner	postgres
Kind	infix
Left type	geometry
Right type	geometry
Result type	bool
Operator function	geometry_overlap
Commutator	<&&
Negator	
Join function	postgis_gist_joinsel
Restrict function	postgis_gist_sel
Left Sort operator	
Right Sort operator	

```
-- Operator: <&&(geometry, geometry)
-- DROP OPERATOR <&&(geometry, geometry);

CREATE OPERATOR <&&{
    PROCEDURE = geometry_overlap,
    LEFTARG = geometry,
    RIGHTARG = geometry,
    COMMUTATOR = <&&,
    RESTRICT = postgis_gist_sel,
    JOIN = postgis_gist_joinsel};
```

PostGIS Räumliche Funktionen

Ansicht in pgAdmin III

The screenshot displays the pgAdmin III interface. On the left, the 'Object browser' shows a tree of functions, with 'addbbox(geometry)' selected. The right pane is divided into two sections: 'Properties' and 'SQL pane'.

Properties:

Property	Value
Name	addbbox
OID	17898
Owner	postgres
Argument count	1
Arguments	geometry
Return type	geometry
Language	c
Returns a set?	No
Object file	\$libdir/liblwgeom.dll
Link symbol	LWGEOM_addBBOX
Volatility	IMMUTABLE
Security of definer?	No
Strict?	Yes
ACL	
System function?	No
Comment	

SQL pane:

```
-- Function: addbbox(geometry)
-- DROP FUNCTION addbbox(geometry);

CREATE OR REPLACE FUNCTION addbbox(geometry)
  RETURNS geometry AS
  '$libdir/liblwgeom.dll', 'LWGEOM_addBBOX'
  LANGUAGE 'c' IMMUTABLE STRICT;
ALTER FUNCTION addbbox(geometry) OWNER TO postgres;
```

Retrieving Function details... Done. 0.00 secs

PostGIS Systemtabellen

spatial_ref_sys – Liste der Projektionen mit OGC SRTText und proj4 Parametern

	srid [PK] integer	auth_name character varying	auth_srid integer	srttext character varying(2048)	proj4text character varying(2048)
1931	21483	EPSG	21483	PROJCS["Beijing 1954 / Gauss-Kruger 23N (deprecated)",GEOGCS["Beijing 1954",DATUM["Beijing_1954",SPHERO	+proj=tmerc +lat_0=0 +lon_0=135 +k=1.000000 +x_0=500000 +y_0=0 +ellps=krass
1932	21500	EPSG	21500	PROJCS["Belge 1950 (Brussels) / Belge Lambert 50",GEOGCS["Belge 1950 (Brussels)",DATUM["Reseau_National_f	+proj=lcc +lat_1=49.83333333333334 +lat_2=51.16666666666666 +lat_0=90 +lon_0
1933	21780	EPSG	21780	PROJCS["Bern 1898 (Bern) / LV03C",GEOGCS["Bern 1898 (Bern)",DATUM["CH1903_Bern",SPHEROID["Bessel 184	+proj=somerc +lat_0=46.95240555555556 +lon_0=0 +x_0=0 +y_0=0 +ellps=bessel
1934	21781	EPSG	21781	PROJCS["CH1903 / LV03",GEOGCS["CH1903",DATUM["CH1903",SPHEROID["Bessel 1841",6377397.155,299.152]	+proj=somerc +lat_0=46.95240555555556 +lon_0=7.439583333333333 +x_0=600000
1935	21817	EPSG	21817	PROJCS["Bogota 1975 / UTM zone 17N (deprecated)",GEOGCS["Bogota 1975",DATUM["Bogota_1975",SPHEROID["	+proj=utm +zone=17 +ellps=intl +towgs84=307,304,-318,0,0,0,0 +units=m +no_def
1936	21818	EPSG	21818	PROJCS["Bogota 1975 / UTM zone 18N",GEOGCS["Bogota 1975",DATUM["Bogota_1975",SPHEROID["Internationa	+proj=utm +zone=18 +ellps=intl +towgs84=307,304,-318,0,0,0,0 +units=m +no_def
1937	21891	EPSG	21891	PROJCS["Bogota 1975 / Colombia West zone (deprecated)",GEOGCS["Bogota 1975",DATUM["Bogota_1975",SPHE	+proj=tmerc +lat_0=4.599047222222222 +lon_0=-77.08091666666667 +k=1.000000
1938	21892	EPSG	21892	PROJCS["Bogota 1975 / Colombia Bogota zone (deprecated)",GEOGCS["Bogota 1975",DATUM["Bogota_1975",SP	+proj=tmerc +lat_0=4.599047222222222 +lon_0=-74.08091666666667 +k=1.000000
1939	21893	EPSG	21893	PROJCS["Bogota 1975 / Colombia East Central zone (deprecated)",GEOGCS["Bogota 1975",DATUM["Bogota_197	+proj=tmerc +lat_0=4.599047222222222 +lon_0=-71.08091666666667 +k=1.000000
1940	21894	EPSG	21894	PROJCS["Bogota 1975 / Colombia East (deprecated)",GEOGCS["Bogota 1975",DATUM["Bogota_1975",SPHEROID	+proj=tmerc +lat_0=4.599047222222222 +lon_0=-68.08091666666667 +k=1.000000
1941	21896	EPSG	21896	PROJCS["Bogota 1975 / Colombia West zone",GEOGCS["Bogota 1975",DATUM["Bogota_1975",SPHEROID["Interr	+proj=tmerc +lat_0=4.599047222222222 +lon_0=-77.08091666666667 +k=1.000000
1942	21897	EPSG	21897	PROJCS["Bogota 1975 / Colombia Bogota zone",GEOGCS["Bogota 1975",DATUM["Bogota_1975",SPHEROID["Inte	+proj=tmerc +lat_0=4.599047222222222 +lon_0=-74.08091666666667 +k=1.000000
1943	21898	EPSG	21898	PROJCS["Bogota 1975 / Colombia East Central zone",GEOGCS["Bogota 1975",DATUM["Bogota_1975",SPHEROID	+proj=tmerc +lat_0=4.599047222222222 +lon_0=-71.08091666666667 +k=1.000000
1944	21899	EPSG	21899	PROJCS["Bogota 1975 / Colombia East",GEOGCS["Bogota 1975",DATUM["Bogota_1975",SPHEROID["Internationa	+proj=tmerc +lat_0=4.599047222222222 +lon_0=-68.08091666666667 +k=1.000000
1945	22032	EPSG	22032	PROJCS["Camacupa / UTM zone 32S",GEOGCS["Camacupa",DATUM["Camacupa",SPHEROID["Clarke 1880 (RGS)",	+proj=utm +zone=32 +south +ellps=clrk80 +units=m +no_defs
1946	22033	EPSG	22033	PROJCS["Camacupa / UTM zone 33S",GEOGCS["Camacupa",DATUM["Camacupa",SPHEROID["Clarke 1880 (RGS)",	+proj=utm +zone=33 +south +ellps=clrk80 +units=m +no_defs
1947	22091	EPSG	22091	PROJCS["Camacupa / TM 11.30 SE",GEOGCS["Camacupa",DATUM["Camacupa",SPHEROID["Clarke 1880 (RGS)",6	+proj=tmerc +lat_0=0 +lon_0=11.5 +k=0.999600 +x_0=500000 +y_0=10000000 +e
1948	22092	EPSG	22092	PROJCS["Camacupa / TM 12 SE",GEOGCS["Camacupa",DATUM["Camacupa",SPHEROID["Clarke 1880 (RGS)",6376	+proj=tmerc +lat_0=0 +lon_0=12 +k=0.999600 +x_0=500000 +y_0=10000000 +ellp
1949	22171	EPSG	22171	PROJCS["POSGAR 98 / Argentina 1",GEOGCS["POSGAR 98",DATUM["Posiciones_Geodesicas_Argentinas_1998",S	+proj=tmerc +lat_0=-90 +lon_0=-72 +k=1.000000 +x_0=1500000 +y_0=0 +ellps=GRS

3162 rows.

PostGIS Geometrie-Datentypen

PostGIS EWKT/EWKB: 3D, 4D, M-Werte, etc.

SQL-MM Part 3 (tw. Implementiert): div. Kurven, Surface, etc.

OpenGIS Simple Features for SQL (WKB, WKT)

Legende:

WKT: Well-Known Text (OGC)

WKB: Well-Known Binary (OGC)

EWKT/EWKB: Extended WKT und WKB (PostGIS Spezifisch); plus 3d + plus M-Werte

SQL-MM: SQL Multimedia Applications Spatial specification: Curve Extensions

Beispiele für Simple Features

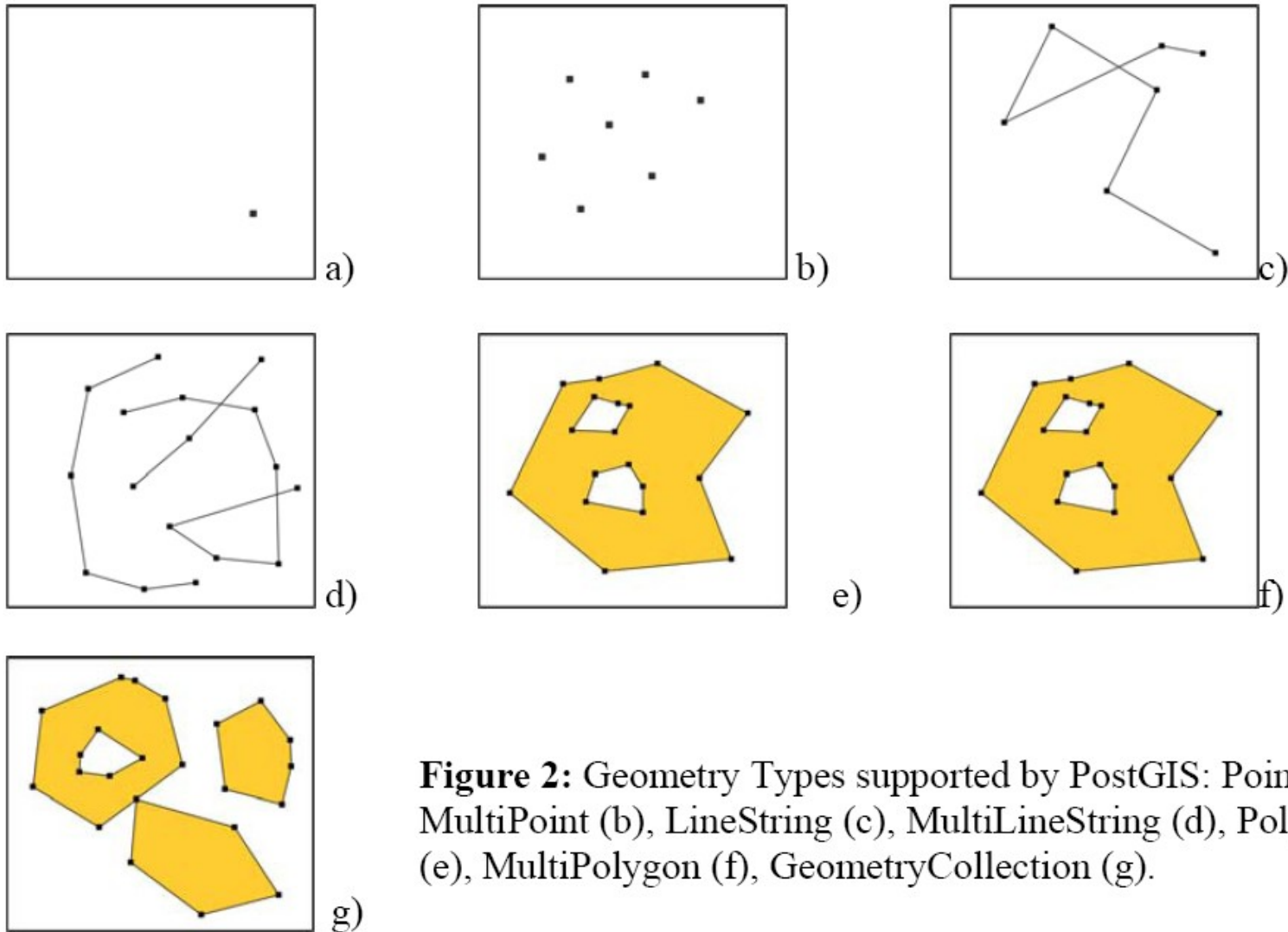


Figure 2: Geometry Types supported by PostGIS: Point (a), MultiPoint (b), LineString (c), MultiLineString (d), Polygon (e), MultiPolygon (f), GeometryCollection (g).

Beispiele f. WKT Geometrien (Simple Features)

`POINT(0 0)`

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

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

`MULTIPOINT(0 0,1 2)`

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

`MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)),
((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))`

`GEOMETRYCOLLECTION(POINT(2 3),LINESTRING((2 3,3 4)))`

Beispiele PostGIS EWKT Geometrien

```
POINT(0 0 0) -- XYZ
```

```
SRID=32632;POINT(0 0) -- XY with SRID
```

```
POINTM(0 0 0) -- XYM
```

```
POINT(0 0 0 0) -- XYZM
```

```
SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- XYM with SRID
```

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

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

```
MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2  
2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1  
0)))
```

```
GEOMETRYCOLLECTIONM(POINTM(2 3 9), LINESTRINGM(2 3 4, 3 4 5))
```

SQL MM Part 3 Geometrien

```
CIRCULARSTRING(0 0, 1 1, 1 0)
```

```
COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0), (1 0, 0 1))
```

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

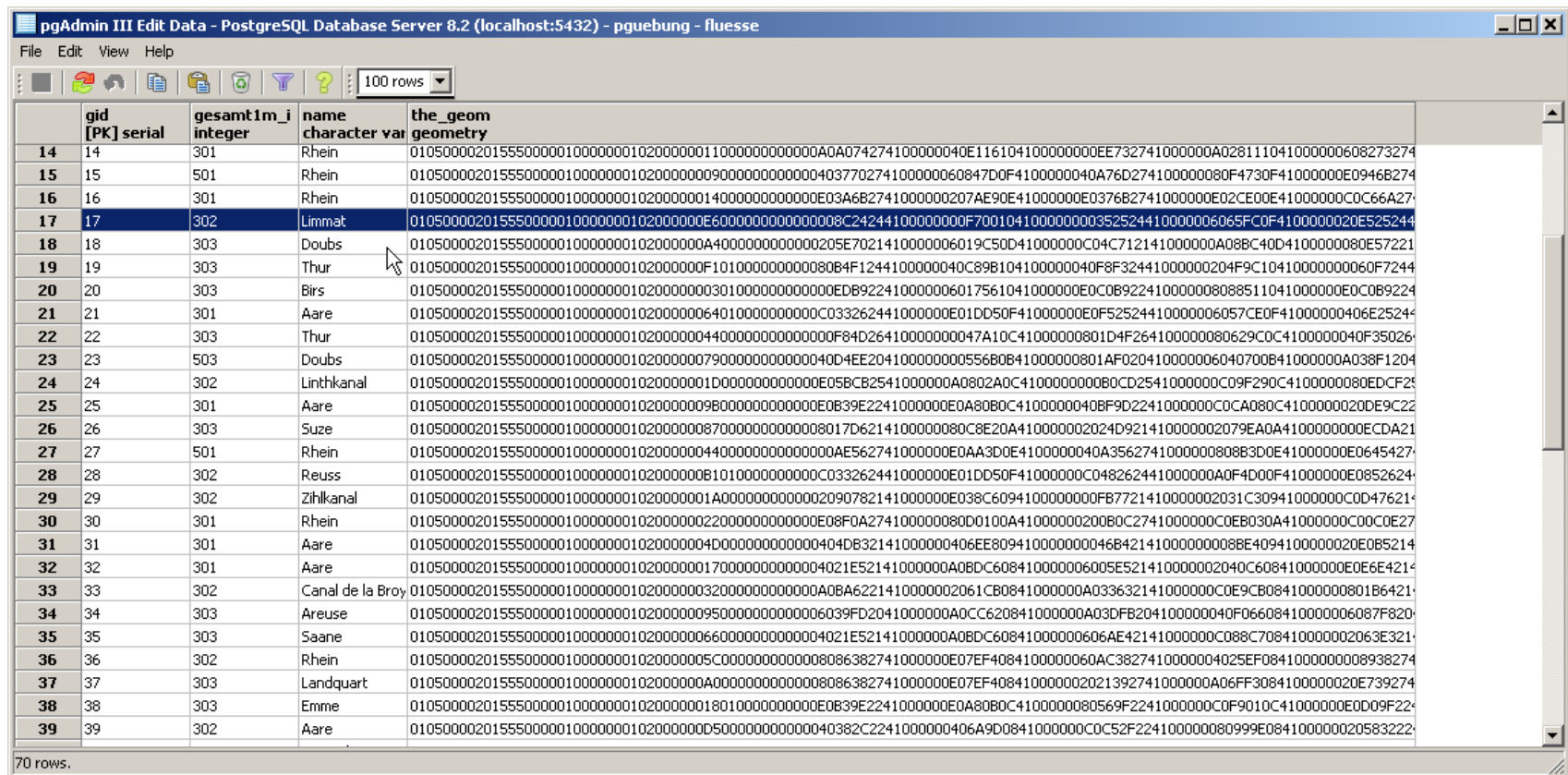
```
MULTICURVE((0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4))
```

```
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)))
```

```
GEOMETRYCOLLECTIONM(POINTM(2 3 9), LINESTRINGM(2 3 4, 3 4 5))
```

PostGIS internes Speicherformat: LWGEOM

Internes Speicherformat LWGEOM (Lightweight Geometry) aber Default Repräsentation bei Queries ist EWKB



	gid [PK] serial	gesamt1m_i integer	name character var	the_geom geometry
14	14	301	Rhein	01050000201555000001000000010200000011000000000000A0A074274100000040E116104100000000EE732741000000A028111041000000608273274
15	15	501	Rhein	01050000201555000001000000010200000090000000000000403770274100000060847D0F4100000040A76D274100000080F4730F41000000E0946B274
16	16	301	Rhein	01050000201555000001000000010200000014000000000000E03A682741000000207AE90E41000000E037682741000000E02CE00E41000000C0C66A274
17	17	302	Limmat	010500002015550000010000000102000000E6000000000000008C24244100000000F700104100000000352524410000006065FC0F4100000020E525244
18	18	303	Doubs	010500002015550000010000000102000000A4000000000000205E7021410000006019C50D41000000C04C712141000000A08BC40D4100000080E57221
19	19	303	Thur	010500002015550000010000000102000000F101000000000080B4F1244100000040C89B104100000040F8F32441000000204F9C104100000000607F244
20	20	303	Birs	0105000020155500000100000001020000003010000000000000EDB922410000006017561041000000E0C0B922410000008088511041000000E0C0B9224
21	21	301	Aare	01050000201555000001000000010200000064010000000000C033262441000000E01DD50F41000000E0F52524410000006057CE0F41000000406E25244
22	22	303	Thur	0105000020155500000100000001020000004400000000000000F84D26410000000047A10C41000000801D4F264100000080629C0C4100000040F350264
23	23	503	Doubs	0105000020155500000100000001020000007900000000000040D4EE20410000000055680B41000000801AF020410000006040700B41000000A038F1204
24	24	302	Linthkanal	0105000020155500000100000001020000001D000000000000E05BCB2541000000A0802A0C410000000080CD2541000000C09F290C4100000080EDCF254
25	25	301	Aare	0105000020155500000100000001020000009B000000000000E0B39E2241000000E0A80B0C4100000040BF9D2241000000C0CA080C4100000020DE9C224
26	26	303	Suze	010500002015550000010000000102000000870000000000008017D6214100000080C8E20A410000002024D921410000002079EA0A4100000000ECD4214
27	27	501	Rhein	0105000020155500000100000001020000004400000000000000AE562741000000E0AA3D0E4100000040A3562741000000808B3D0E41000000E06454214
28	28	302	Reuss	010500002015550000010000000102000000810000000000C033262441000000E01DD50F41000000C048262441000000A0F4D00F41000000E08526244
29	29	302	Zihlkanal	0105000020155500000100000001020000001A000000000000002090782141000000E038C6094100000000FB7721410000002031C30941000000C0D476214
30	30	301	Rhein	01050000201555000001000000010200000022000000000000E08F0A274100000080D0100A41000000200B0C2741000000C0EB030A41000000C00C0E274
31	31	301	Aare	0105000020155500000100000001020000004D000000000000404DB32141000000406EE809410000000046842141000000008BE4094100000020E0B5214
32	32	301	Aare	010500002015550000010000000102000000170000000000004021E52141000000A0BD6C608410000006005E521410000002040C60841000000E0E6E4214
33	33	302	Canal de la Broye	01050000201555000001000000010200000032000000000000A0BA6221410000002061CB0841000000A033632141000000C0E9CB0841000000801B64214
34	34	303	Areuse	010500002015550000010000000102000000950000000000006039FD2041000000A0CC620841000000A03DFB204100000040F06608410000006087F8204
35	35	303	Saane	010500002015550000010000000102000000660000000000004021E52141000000A0BD6C60841000000606AE42141000000C088C708410000002063E3214
36	36	302	Rhein	0105000020155500000100000001020000005C0000000000008086382741000000E07EF4084100000060AC3827410000004025EF0841000000008938274
37	37	303	Landquart	010500002015550000010000000102000000A00000000000008086382741000000E07EF408410000002021392741000000A06FF3084100000020E739274
38	38	303	Emme	01050000201555000001000000010200000018010000000000E0B39E2241000000E0A80B0C4100000080569F2241000000C0F9010C41000000E0D09F224
39	39	302	Aare	010500002015550000010000000102000000D500000000000040382C2241000000406A9D0841000000C0C52F224100000080999E0841000000205832224

Datenbank räumlich einrichten

Methode 1 (copy/paste):

- Bestehendes leeres räumliches Template kopieren:

```
CREATE DATABASE my_spatial_db  
TEMPLATE=template_postgis
```

Methode 2 (manuell):

- Datenbank kreieren in shell:

```
createdb  
my_spatial_db
```
- PL/PGSQL aktivieren in shell:

```
createlang  
plpgsql my_spatial_db
```
- PostGIS Objekte und Funktionen laden:

```
psql -d  
my_spatial_db -f lwpostgis.sql
```
- Koordinatensystemsdefinitionen laden:

```
psql -d  
my_spatial_db -f spatial_ref_sys.sql
```

Tabelle räumlich einrichten

Zuerst normale Tabelle erstellen:

- `CREATE TABLE eisenbahnen (ID int4, NAME varchar(25))`

Dann räumliche Spalte hinzufügen

- `SELECT AddGeometryColumn('public', 'eisenbahnen', 'geom', 21781, 'MULTILINESTRING', 2)`

Bestehende räumliche Daten laden

- SQL Datei laden
- ESRI shapefile laden: shape2pgsql, danach SQL File laden mit pgsq
- ogr2ogr
- FME
- Über GIS Systeme: QGIS, JumpGIS, GRASS, etc.
- DUMP and RESTORE von einer anderen Postgis-Datenbank, auch über pgAdmin

ESRI Shapefile Loader

- ESRI Shapefile ins SQL Format konvertieren:

```
shp2pgsql [<options>] <shapefile>  
[<schema>.]<table> > <sqlfile>
```

z.B.

```
shp2pgsql -s 21781 -I -W ISO-8859-1 G3G07  
public.gemeinden >gemeinden.sql
```

- SQL File importieren:

```
psql [OPTIONS]... [DBNAME [USERNAME]]
```

z.B.

```
psql -d dbname -U username -f gemeinden.sql
```

ogr2ogr (verschiedene Formate)

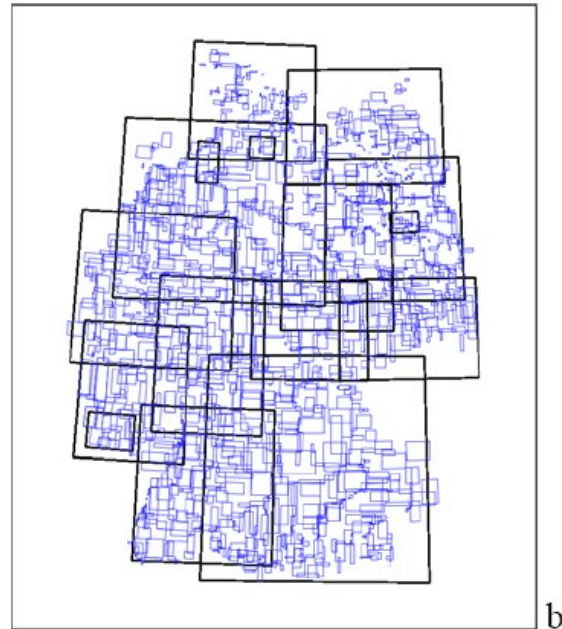
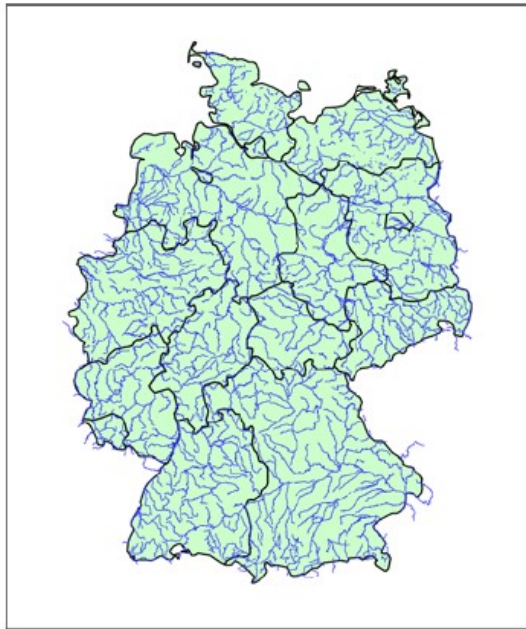
```
Usage: ogr2ogr [-skipfailures] [-append] [-update] [-f format_name]
      [-select field_list] [-where restricted_where]
      [-sql <sql statement>] [--help-general]
      [-spat xmin ymin xmax ymax] [-preserve_fid] [-fid FID]
      [-a_srs srs_def] [-t_srs srs_def] [-s_srs srs_def]
      [[-dsco NAME=VALUE] ...] dst_datasource_name src_datasource_name
      [-lco NAME=VALUE] [-nln name] [-nlt type] [layer [layer ...]]
```

z.B Interlis 1 File nach Postgis laden:

```
ogr2ogr -f PostgreSQL PG:dbname=warmerda
av_fixpunkte_ohne_LFPNachfuehrung.itf,av.ili
-lco OVERWRITE=yes
```


Räumliche Indizes Erstellen

- PostGIS räumliche Indizes basieren auf R-Trees, implementiert mit der GIST Infrastruktur und beschleunigen die Abfragen deutlich
- R-Trees organisieren die Boundingboxen der Daten in verschachtelte räumliche Rechtecke



Räumliche Indizes Erstellen, Beispiel

Syntax:

```
CREATE INDEX [indexname] ON [tablename] USING  
    GIST ( [geometryfield] GIST_GEOMETRY_OPS );
```

Beispiel:

```
CREATE INDEX in_the_geom_gist ON eisenbahnen  
    USING GIST ( the_geom GIST_GEOMETRY_OPS );
```

Räumliche Indizes bei Abfragen nutzen

Achtung: die Indizes werden nur im Zusammenhang mit den bounding-box basierten Operatoren (z.b. && (bounding box overlap)) verwendet!

Beispiel:

```
SELECT the_geom FROM geom_table WHERE the_geom
    && 'BOX3D(90900 190900, 100100 200100)'::box3d
AND distance( the_geom,
    GeomFromText( 'POINT(100000 200000)', -1 )) < 100
```

Daher: nach Möglichkeit mindestens einen bounding-box Operator (z.B. &<,&&,&>,<<,>>,etc.) verwenden!

Siehe cheatsheet (BostonGIS), für weitere Funktionen die den Index verwenden

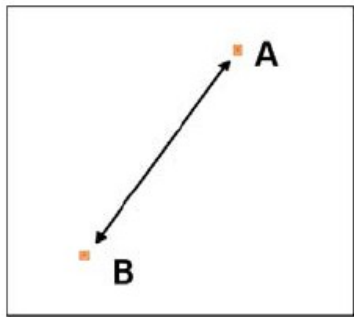
Räumliche Funktionen und Operatoren

- Data Management Functions (Datenmanagement)
- Geometry Relationship Functions (Beziehungen)
- Geometry Processing Functions (Berechnungen)
- Geometry Accessors (Zugriffsfunktionen)
- Geometry Constructors (Geometrie erzeugen)
- Measurement Functions (Messfunktionen)
- Linear Referencing (Berechnungen entlang einer Linie)
- SQL-MM spezifische Funktionen
- ArcSDE spezifische Funktionen

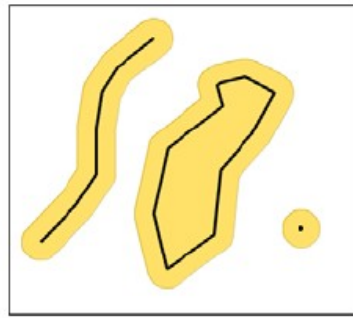
Beispiele für Beziehungsfunktionen

- ST_Distance(geometry, geometry)
- ST_DWithin(geometry, geometry, float)
- ST_Equals(geometry, geometry)
- ST_Disjoint(geometry, geometry)
- ST_Intersects(geometry, geometry)
- ST_Touches(geometry, geometry)
- ST_Crosses(geometry, geometry)
- ST_Within(geometry A, geometry B)
- ST_Overlaps(geometry, geometry)
- ST_Contains(geometry A, geometry B)
- ST_Covers(geometry A, geometry B)
- ST_CoveredBy(geometry A, geometry B)
- ST_Intersects(geometry, geometry)

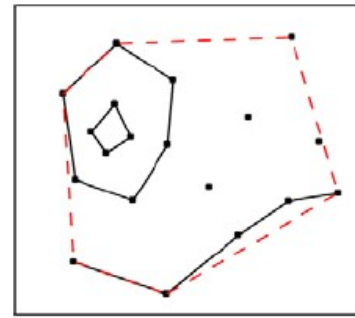
Einige räuml. Beziehungen graphisch dargestellt



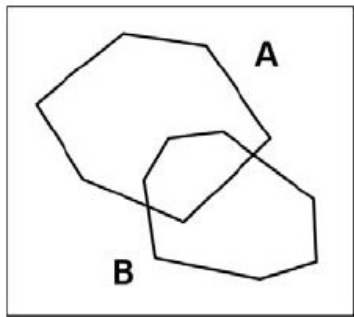
a)



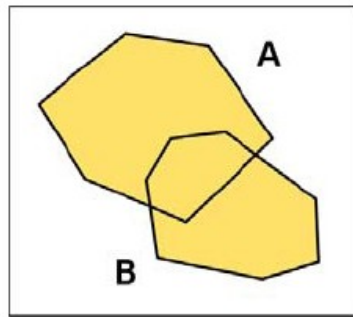
b)



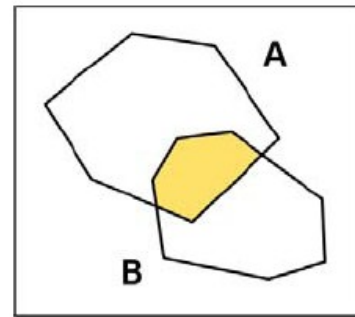
c)



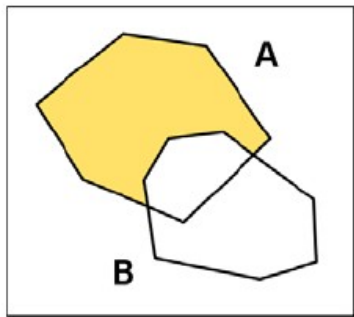
d)



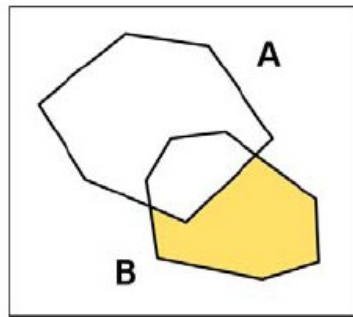
e)



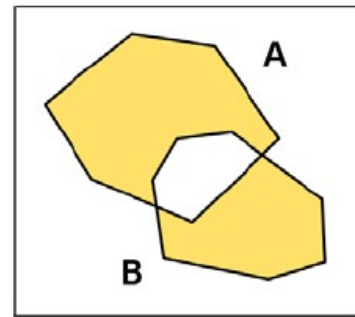
f)



g)



h)



i)

Beispiel „Crosses“

Prüfe ob d. Fluss Emme d. Gemeindegeometrie v. Burgdorf durchläuft

```
SELECT ST_Crosses((SELECT the_geom FROM
gemeinden WHERE name = 'Burgdorf'),
(SELECT the_geom FROM fluesse WHERE name
= 'Emme'));
```

Ergebnis: „t“ (true)

```
SELECT ST_Crosses((SELECT the_geom FROM
gemeinden WHERE name = 'Basel'), (SELECT
the_geom FROM fluesse WHERE name =
'Emme'));
```

Ergebnis: „f“ (false)

Beispiel „Crosses“, Spezialfall Mehrere MULTILINESTRINGs

Prüfe ob die kombinierten Einzel-Linienelemente des Rheinflusses die Gemeindegeometrie von Basel durchläuft

```
SELECT ST_Crosses((SELECT the_geom FROM
gemeinden WHERE name = 'Basel'), (SELECT
ST_Geomunion(the_geom) FROM fluesse
WHERE name = 'Rhein' GROUP BY name));
```

Ergebnis: „t“ (true)

Beispiel 2 „Crosses“

```
SELECT g.name FROM gemeinden g, fluesse f
WHERE f.name = 'Emme' AND
f.the_geom && g.the_geom AND
ST_Crosses(f.the_geom, g.the_geom)
ORDER by name ASC;
```

Ergebnis: Aefligen, Biberist, Burgdorf,
etc.

Beziehungsfunktionen, Spezialfall ST_Relate

- `ST_Relate(geometry, geometry, intersectionPatternMatrix)`

Retourniert ob die räumliche DE-9IM Beziehungsmatrix zweier Geometrien „true“ ergibt

- `ST_Relate(geometry, geometry)`

Retourniert die räumliche Beziehungsmatrix zweier Geometrien

DE-9IM = Dimensionally Extended Nine-Intersection Model

Beispiele für Geometry Processing Functions

- ST_Centroid(geometry)
- ST_Area(geometry)
- ST_Length(geometry)
- ST_PointOnSurface(geometry)
- ST_Boundary(geometry)
- ST_Buffer(geometry, double, [integer])
- ST_ConvexHull(geometry)
- ST_Intersection(geometry, geometry)
- ST_SymDifference(geometry A, geometry B)
- ST_Difference(geometry A, geometry B)
- ST_Union(geometry, geometry)
- ST_Union(geometry set)
- ST_MemUnion(geometry set)

Beispiele „Area“, „Length“

```
SELECT ST_Area(the_geom) FROM gemeinden  
WHERE name = 'Uster';
```

Ergebnis: 28799569.5 m2
(Achtung: generalisierte Geometrie!)

```
SELECT Sum(ST_Length(the_geom)) FROM  
fluesse WHERE name = 'Aare';
```

Ergebnis: 191989.418949
(Achtung: generalisierte Geometrie!)

Beispiel Buffer() und Within()

Selektiere Städte der Schweiz die sich innerhalb eines Buffers von 10km um den Fluss Emme befinden

```
SELECT "FULL NAME" FROM staedte schweiz
WHERE ST_Within(the_geom, (SELECT
ST_Buffer(the_geom, 10000) FROM fluesse
WHERE name = 'Emme')) ORDER BY
"FULL_NAME" ASC;
```

Resultat:

Aedermannsdorf, Aelgäu, Aeschau,
Affoltern, ...

Beispiel GEOMUnion()

Fasse alle Gemeinden die die gleiche Kantons-Id haben zusammen und schreibe sie in eine neue Tabelle

Zuerst: Neue Tabelle erstellen (Primary Key auf Spalte kt von type integer)

```
SELECT AddGeometryColumn('public',  
    'kantone', 'the_geom', 21781,  
    'MULTIPOLYGON', -2);  
INSERT INTO kantone SELECT  
    kt, ST_Multi(geomunion(the_geom)) FROM  
    gemeinden GROUP BY kt;
```

Danach visuelle Überprüfung im Viewer

Geometry Output Functions

- ST_AsBinary(geometry, {'NDR'|'XDR'})
- ST_AsEWKT(geometry)
- ST_AsEWKB(geometry, {'NDR'|'XDR'})
- ST_AsHEXEWKB(geometry, {'NDR'|'XDR'})
- ST_AsSVG(geometry, [rel], [precision])
- ST_AsGML([version], geometry, [precision])
- ST_AsKML(geometry, [precision])

Nützlich: Simplify() und Intersect() zum Generalisieren und Ausstanzen

Beispiel SVG und KML Output

```
SELECT  
AsSVG(ST_Multi(geomunion(the_geom)), 1, 1)  
FROM fluesse GROUP BY name;
```

```
SELECT  
AsKML(ST_Multi(geomunion(the_geom)), 6)  
FROM fluesse GROUP BY name;
```

OUTPUT ist nicht fertig! – muss mit Dateiköpfen und XML Strukturen ergänzt werden

Beispiel SVG Applikationen

Yosemite Nationalpark:

<http://www.carto.net/williams/yosemite/>

Geofoto Schweiz:

<http://www.geofoto.ch/geophotomap/>

Braucht einen SVG fähigen Browser:

Opera9, Safari3, Firefox 2 (besser 3), Internet Explorer
braucht Plugin (Adobe SVG Viewer)

Koordinatentransformationen

- EPSG Codes (European Petroleum Survey Group) = Riesige Auflistung von unkontrollierten Projektionsparametern
- Standardisierte OGC SRID Definitionen
- In PostGIS in der Tabelle `spatial_ref_sys` abgelegt
- Verwendet Proj4 Library
- Input Daten müssen eine SRID (Nicht -1!) zugewiesen haben
- Umprojizieren mit `Transform()` Kommando
- Bei häufigen Abfragen macht das doppelte Ablegen von Geometrien allenfalls Sinn. Ein Record kann mehrere Geometrien haben

Beispiel Umprojizierung

Selektiere die Koordinaten der Städte in der Schweiz im WGS84 Format. Original liegen die Geometrien im CH1903/LV95

```
SELECT AsText(Transform(the_geom, 4326)) FROM  
staedte_schweiz;
```

Resultat:

```
POINT(7.3xxx, 46.5xxx)
```

...