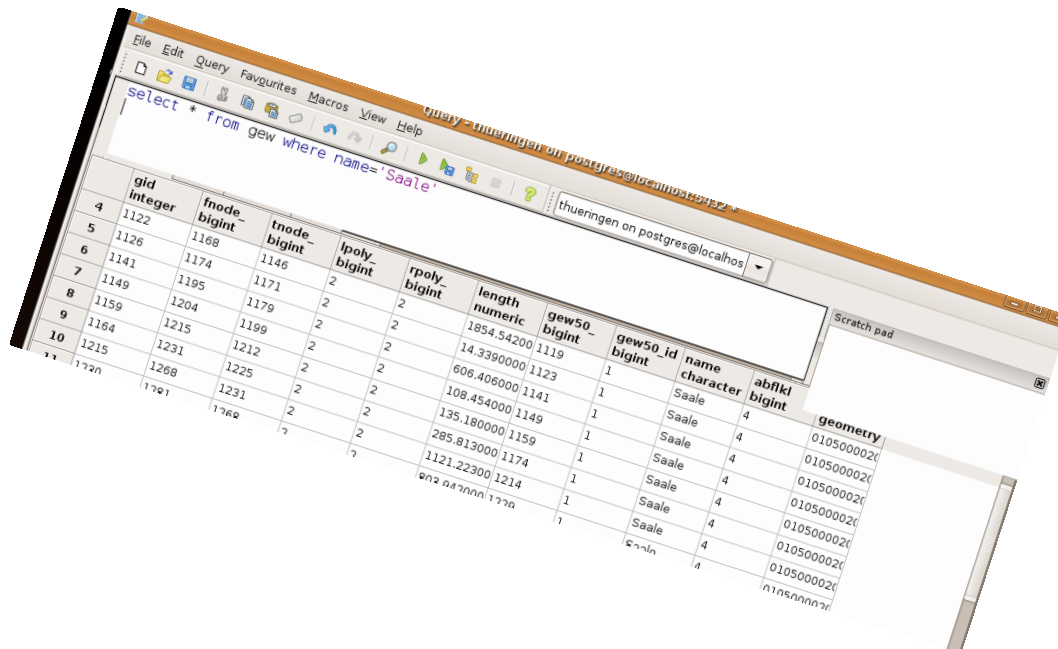


GIS Kurs

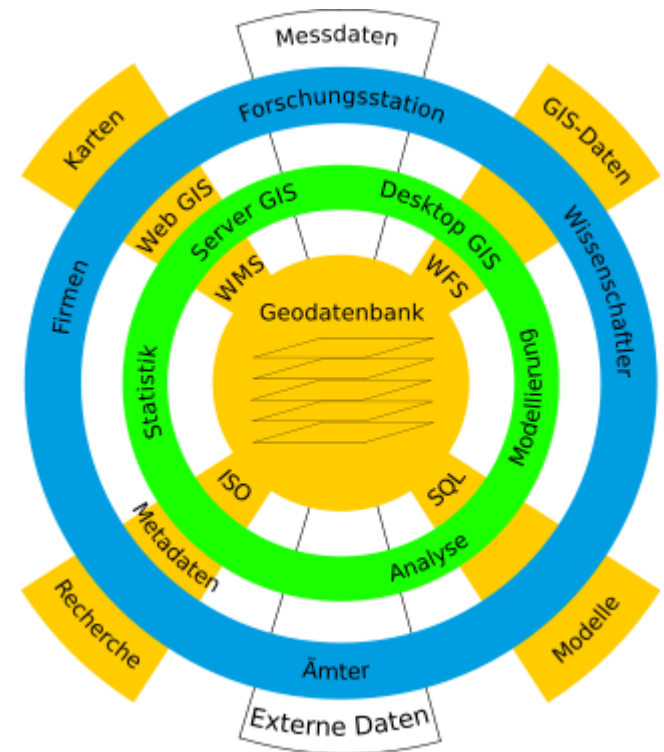
- Geodatenbanken -

Carsten Busch



select * from gew where name='Saale'

gid	integer	lnode_bbigint	lnode_bbigint	lpoly_bbigint	rpoly_bbigint	length	numeric	gew50_bbigint	gew50_bbigint	name	character	abflkt_bbigint	geometry
4	1122	1168	1171	2	2	1854.54200	1119	1	14.33900000	Saale	4	0105000020	
5	1126	1174	1171	2	2	108.4540000	1141	1	135.1800000	Saale	4	0105000020	
6	1141	1195	1179	2	2	285.8130000	1174	1	1121.22300	Saale	4	0105000020	
7	1149	1204	1199	2	2	108.4540000	1149	1	135.1800000	Saale	4	0105000020	
8	1159	1215	1212	2	2	108.4540000	1149	1	135.1800000	Saale	4	0105000020	
9	1164	1231	1225	2	2	108.4540000	1149	1	135.1800000	Saale	4	0105000020	
10	1215	1268	1231	2	2	108.4540000	1149	1	135.1800000	Saale	4	0105000020	



[Kompetenzzentrum für Sensoren und GIS, SENGIS]

Agenda

- Entwicklungstrends bei der Geodatenhaltung
- Architekturen für GIS Programme und Dienste
- Visualisierung von Datenbankinhalten mittels QGIS
- Zugriff auf Datenbanken, SQL als Abfragesprache
 - Einfache Abfragen
 - Gruppierung und Tabellenverknüpfung
 - Unterabfragen und Datenbankviews
- Räumliche SQL Erweiterung (Vektordaten / Rasterdaten)
 - Datenmodell des OpenGeospatial Consortiums (OGC)
 - PostGIS Implementierung
 - Abfragen mittels Spatial SQL
 - Datenexport/-import mit Skripten und QGIS
- Datenmodellierung, Implementierung in PostgreSQL / PostGIS
- Vererbungshierarchien in Tabellenstrukturen
- Datenbanktrigger und PL/SQL

...verwendete Software...

- unbeschränkt verwendbare GIS/DB Komponenten (**Open Source**)
- Vorinstallierte/konfigurierte GIS/DB Umgebungen
- für eigene Problemstellungen einsetzbar
- Software:
 - Virtualisierung: *VM VirtualBox*, Image: *OSGeo*
 - Datenbank: *PostgreSQL/PostGIS*
 - Visualisierung: *QGIS*

CAD/GIS und Datenbanken

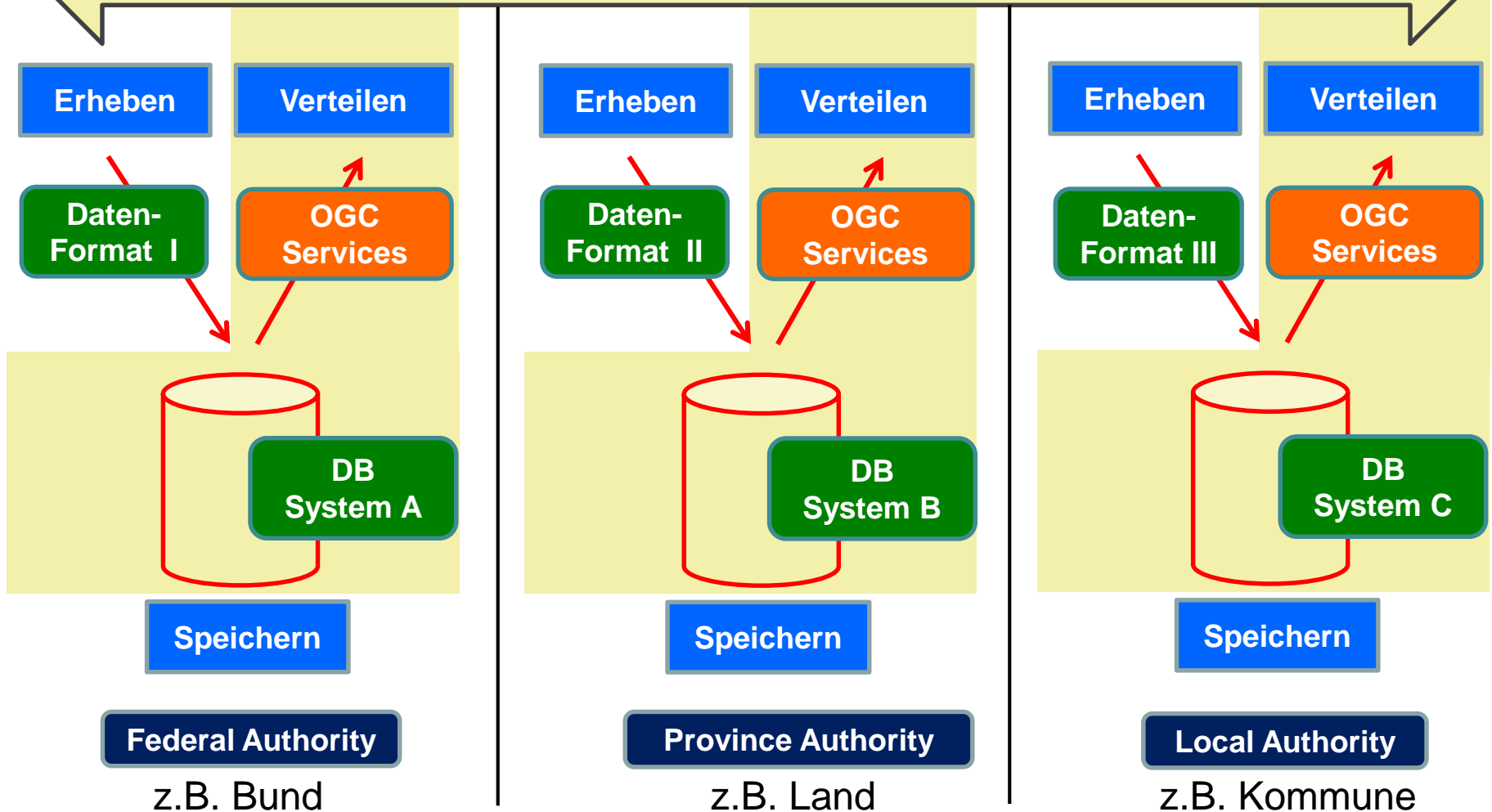
[Geodatenbanken]

- **Trends**

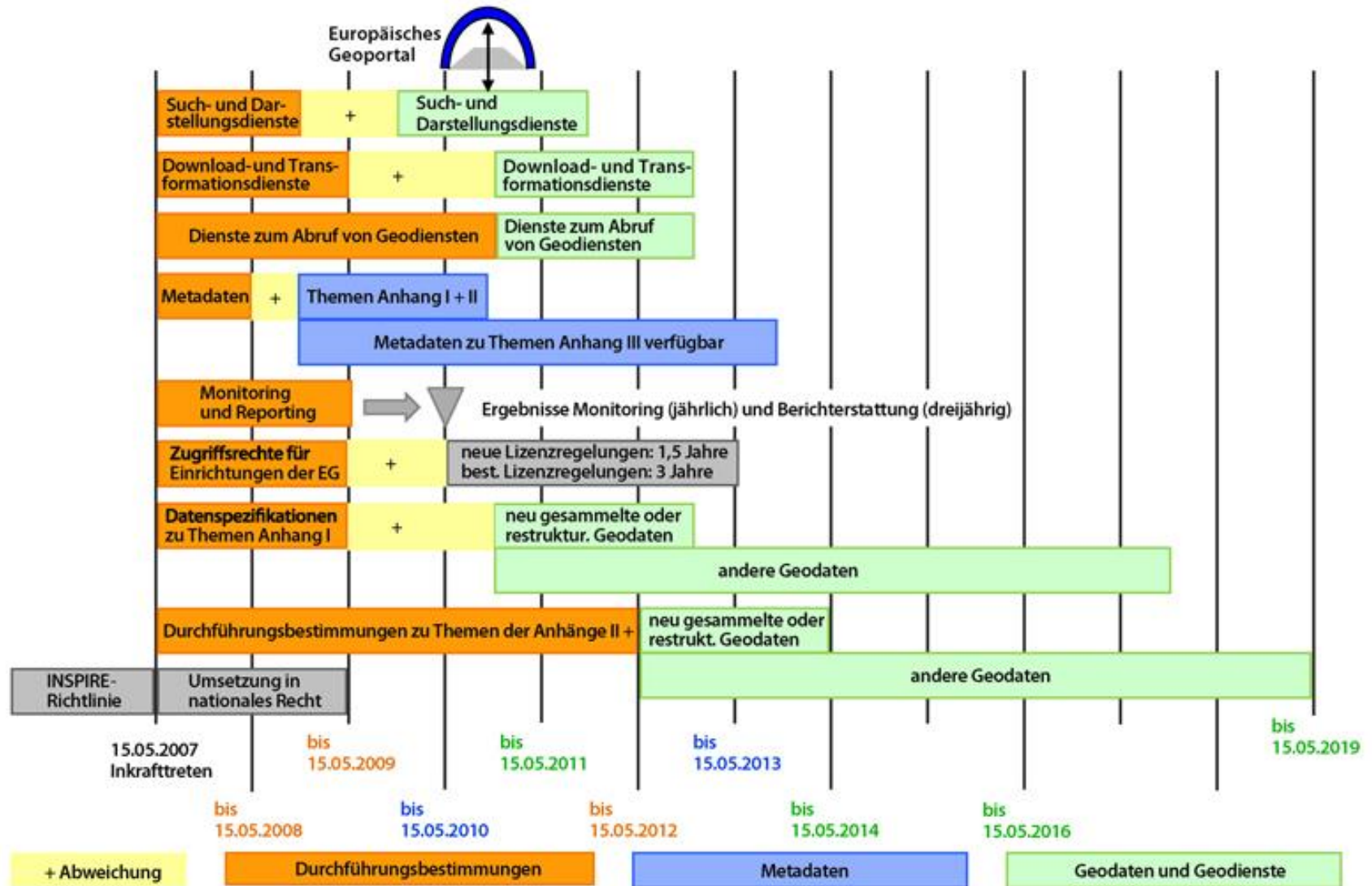
- Gruppenorientierte Entwicklungsarbeit, **gleichzeitige** Bearbeitung einer Zeichnung/GIS-Daten durch **mehrere** Anwender
- Bearbeitung einer Zeichnung/GIS-Daten an **unterschiedlichen** Standorten
- Nutzung des WWW → WebGIS
- Umsetzung nationaler/internationaler Standards
 - **GDI** Geodateninfrastruktur
 - **INSPIRE** Infrastructure for Spatial Information in the European Community
- Übergang vom Datei- zum Datenbanksystem

Konzept der Geodateninfrastrukturen

Skalierbare Geodateninfrastruktur GDI-DE



Zeitplan INSPIRE / GDI-DE



[GDI-DE]

Schnittstellen: Open Geospatial Consortium (OGC) Standards und Dienste

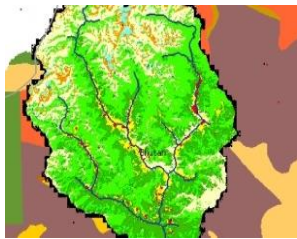


Databases

(e.g. water level information in Oracle, PostgreSQL, IBM DB2,...)



Spatial SQL, SOS
(Sensor Observation Service)

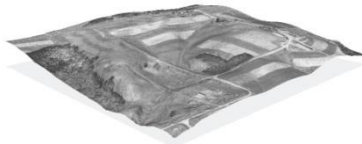


Maps

(e.g. landuse maps in ArcGIS, Mapserver, Open Street Map, ...)



WMS
(Web Map Service)



Raster

(e.g. Digital Elevation Model as GeoTIFF,, JPEG, ...)



WCS
(Web Coverage Service)



Vector

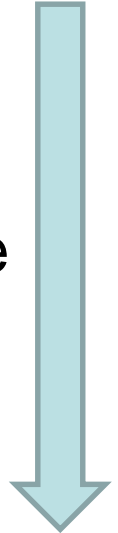
(e.g. street information as Shape, DXF, SVG)



WFS
(Web Feature Service)

Verwendung und Verarbeitung von GIS Daten

- Desktop GIS → Datenerhebung
 - GIS-Analysen mit **lokalen** Daten
 - Ergänzung / Verknüpfung mit **zentralen** Geodaten
 - Verarbeitung **unterschiedlicher** Koordinatensysteme
- Visualisierung / Bereitstellung im Web
→ Datenfreigabe
 - Zugriff über **webbasierte** Dienste
 - Verarbeitung **unterschiedlicher** Koordinatensysteme
 - z.T. **kaskadierende** Strukturen

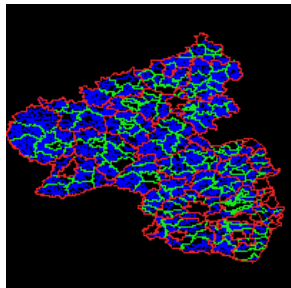


→ Zentralisierung der Geodaten (GeoDB), Zugriff über webbasierter GIS Dienste ←

Daten-/Kartengrundlagen für räumlich basierte (Web-)Informationssysteme

- Eigene Kartensammlungen
 - Digitalisieren
 - Aufnahme im Gelände
- Amtliche (digitale) Kartenwerke (Bundesamt für Kartographie und Geodäsie, Landesvermessungsämter, USGS,...)
 - **ATKIS** Daten, **VG**xxx (Verwaltungsgrenzen), **DLM**xxx (Digitales Landschaftsmodell), **DHM** (Digitales Höhenmodell), **TK**xxx (Topografische Karte)
 - Befliegungsdaten (Luftbilder/Orthophotos)
 - Bereitstellung als Webdienst

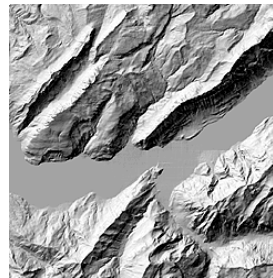
Woher
kommen die
Daten ?



VG Rheinland Pfalz



DLM 250



DHM25



TK25

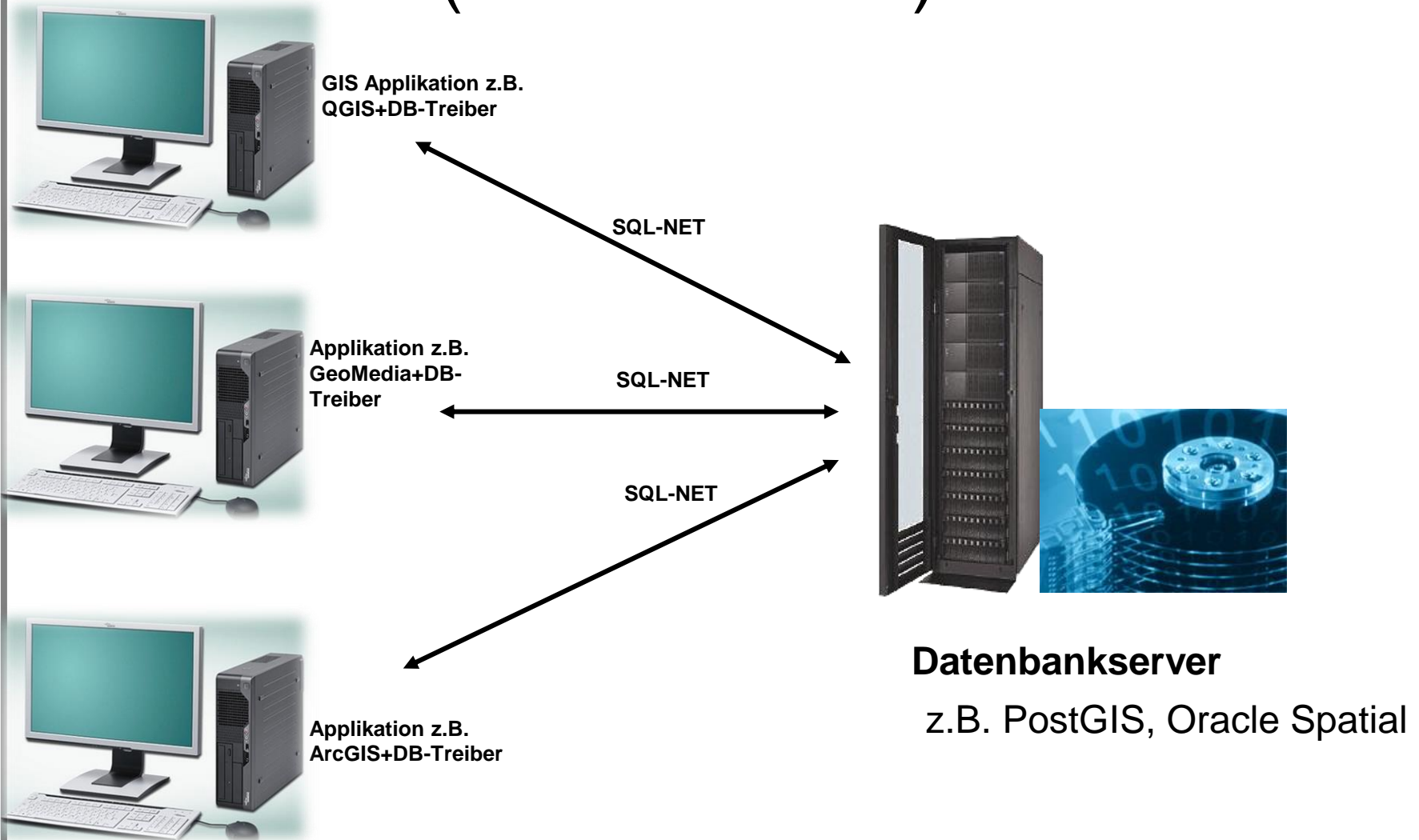
Datenanbieter (ii)

- Kommerzielle Datenanbieter
 - Navteq → Übernahme durch Nokia (2007), enge Kooperation mit Navigon, Garmin → HERE (2011) → Mercedes, Audi, BMW (2015)
 - Teleatlas → Übernahme durch TomTom (2007), enge Kooperation mit Google, Map24, BMW, Daimler AG, Microsoft
- Freie Kartenanbieter

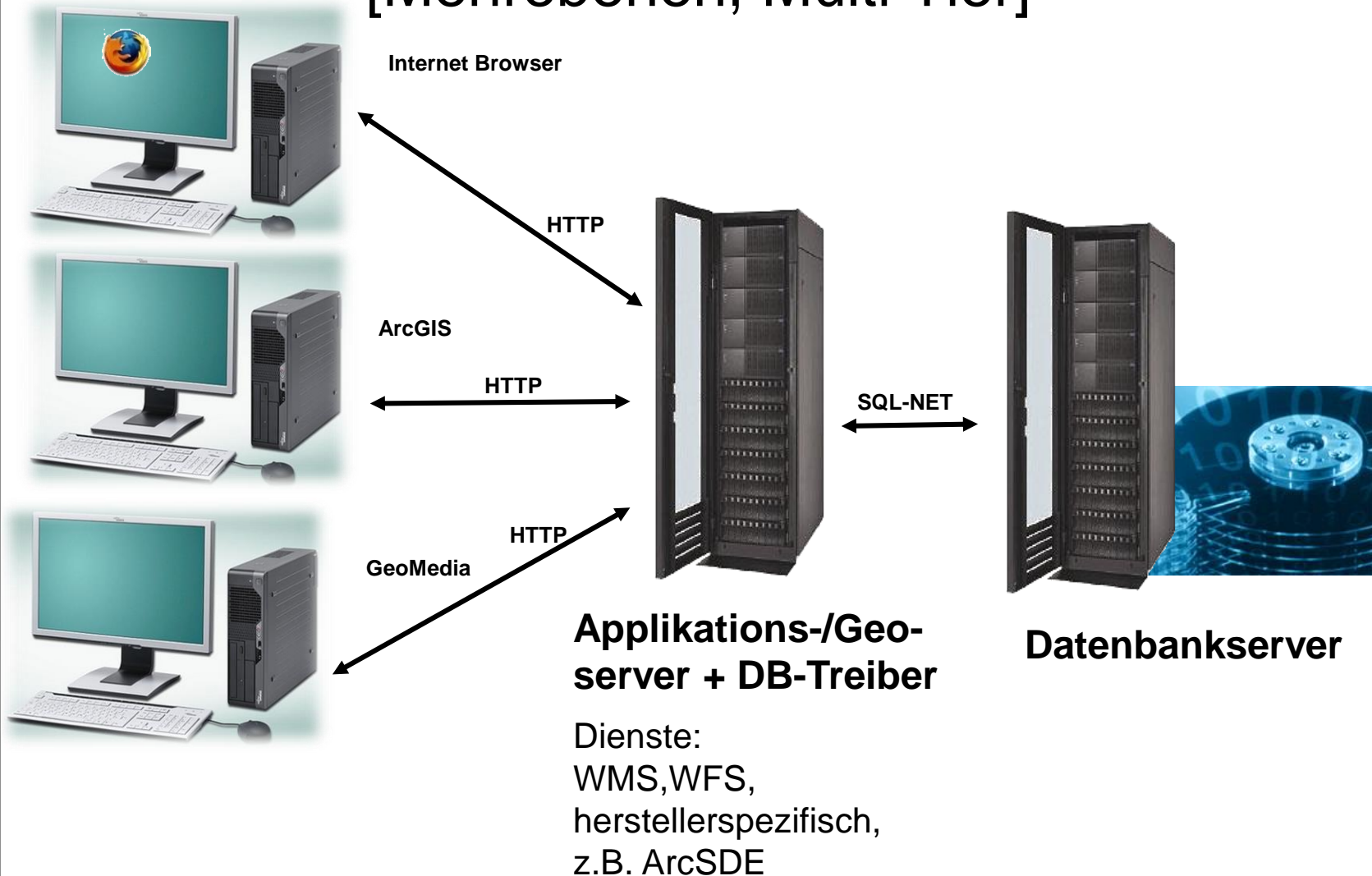


Open Street Map

Zwei Ebenen Architektur (Client-Server)



Applikationsserver-Architektur [Mehrebenen, Multi Tier]



Schichtenmodell in der Softwareentwicklung



Geodatenbanken ...Datenbanken und GIS

- Geometrie- und Sachdaten werden konsistent verwaltet und **unternehmensweit** zur Verfügung gestellt, kontrollierter, **gemeinsamer** Datenzugriff
- Abfrage der Daten nach **Attributwerten** und der **Lage im Raum**
- Große Datenmengen verwaltbar und speicherbar, alphanumerische/räumliche Indizierung, Trennung von Daten und Anwendung
- Backup und Recovery, Versionierung von Datensätzen, Transaktionskonzept
- Abbildung komplexer Datenmodelle (z.B. UML) möglich, z.B. hierarchische Tabellenstrukturen

→ Grundlage für die Verwaltung von Datenbanksystemen: SQL ←

Entwicklung von Datenbanksystemen (DBS)

Relationale Systeme (RDBMS-SQL)

- Seit 1970, für tabellenartige Datenstrukturen, Forschungsprojekt der IBM, System/R (Test mit 8MB großem Datenbestand ;-)) → 1980 DB2 ... → ... 2016 Oracle mit z.T. Datenbeständen im Terrabyte Bereich
- Kontinuierliche Weiterentwicklung der Produkte und Standards



Objektorientierte Systeme (OODBMS)

- Seit 1990, für komplexe Datenstrukturen von CAD und Multimediadaten, z.B. System POET oder Objectivity
- Geringe Verbreitung, Konzepte oftmals in relationalen Systemen direkt oder durch externe Persistenzschichten integriert.



NoSQL Systeme (not only SQL)

- Seit 2000, für datenintensive, verteilte Anwendungen, die auf Atomarität, Konsistenz, Isolation, Dauerhaftigkeit, Transaktionen, etc. (ACID) verzichten können.
- Key Value Ansatz (verteilte Hashtabellen), horizontale Skalierung (mehrere Datenbankserver) durch Webanwendungen wie Facebook, etc.

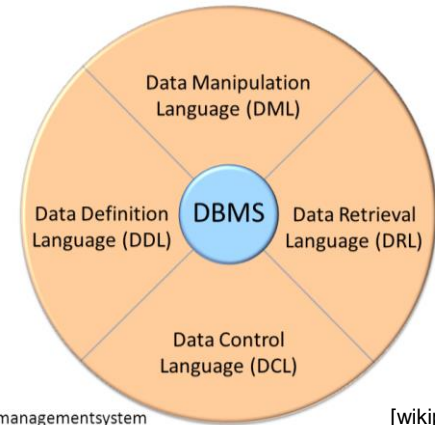
SQL

Sprache zur Abfrage- und Verwaltung von Datenbanken

- SQL: Structured Query Language, Basis System R (IBM)
- 1986 Verabschiedung des ersten Standards **SQL-86**
- 1987 von ISO fast komplett übernommen
- 1992 **SQL**-(9)**2** Referentielle Integrität
- 1999 SQL OGC Erweiterung für Datenbanken
- 2003 **SQL**-200(**3**) Definition von DB Triggern
- 2006 SQL-2006 MultiMedia (MM) SQL und XML, ISO Definition für Geodatenbanken (Spatial Type - **ST**)
- 2011 SQL – 2011, Erweiterung von Triggern, INSTEAD OF für das Aktualisieren von Daten in Sichten
- 2016: Definition in 9 Publikationen, u.a. Anbindung an Java, XML.

Vorteile von SQL

- Einheitliche Datenbanksprache, SQL-3 Standard in vielen Produkten implementiert
- alle wesentlichen Datenbankaktionen werden durch SQL ausgeführt
 - Einfügen / Ändern / Abfrage von Daten
 - Integration datenbankseitiger Programme / Aktionen durch **prozedurales SQL**
 - Nutzerverwaltung
 - Speicherverwaltung (RAM/Festplatten, etc.)



DBMS =
Datenbankmanagementsystem

[wikipedia.org]

Nachteile von SQL / DBMS

- Komplexe, **nichteinheitliche** Syntax
- oft **herstellereigene** SQL-Erweiterungen
- gültige **SQL-Standard bleibt** hinter gegenwärtigen Nutzeranforderungen **zurück**
 - z.B. Partitionierung von Tabellen
- **Aufwendige Migration** zwischen unterschiedlichen Datenbanksystemen

Einordnung von SQL

(generation language GL)

- Programmiersprache der vierten Generation (4-GL)

3.GL (z.B. *Java, C++, Pascal, Basic, ...*)

4.GL

[*Wie*, prozedurale Beschreibung]

Codebeispiel:

```
open(buecher)
while(not eof(buecher))
{
    read(buch)
    if(buch.leihfrist>0)
        print(buch.autor,buch.titel)
}
close(buecher)
```

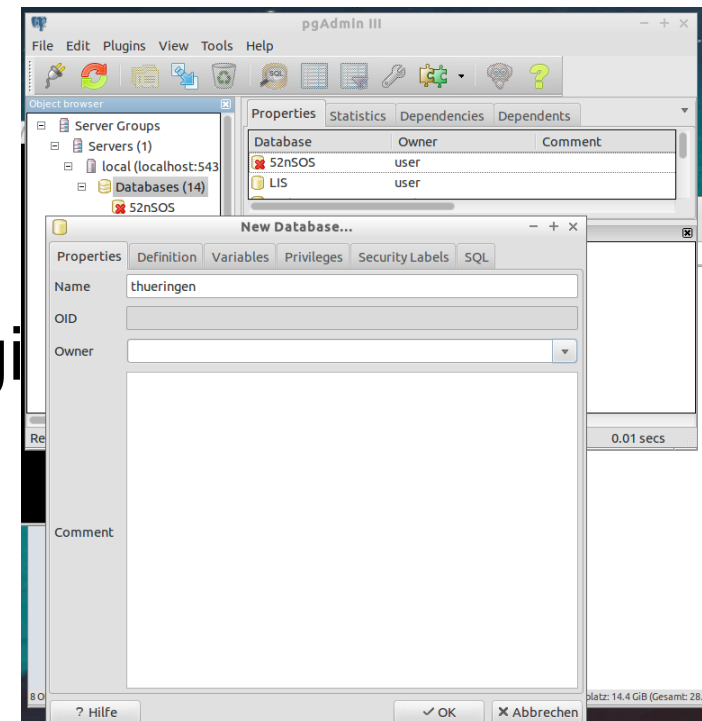
[*Was*, mengenmäßige Beschreibung]

Codebeispiel (SQL)

```
select autor,titel
from buecher
where leihfrist>0
```

Erzeugen einer (Geo-)datenbank

- Angabe auf Kommandozeile oder über Datenbankverwaltungsprogramm (z.B. pgadmin3/4)
- Minimale Angabe
 - CREATE DATABASE *name*
 - CREATE EXTENSION postgis
- Optional:
 - Angabe des Zeichensatzes (UTF-8 oder LATIN9)



Eingabe/Änderung der Daten

- Neuer Datensatz für die Gemeinde Umpferstedt (gkz, name, einwohner)
 - **INSERT INTO** gemeinden **VALUES** (16071089, 'Umpferstedt', 638)
- Veränderung der Einwohneranzahl
 - **UPDATE** gemeinden **SET** einwohner = 620 **WHERE** gkz = 16071089
- Wegfall aufgrund einer Gemeindereform
 - **DELETE FROM** gemeinden **WHERE** gkz = 16071089

Import/Export von Daten

- Dienstprogramme des Datenbankherstellers mit Kommandozeilenaufruf
 - shp2pgsql und pgsql2shp
 - Geeignet für Batchbetrieb und große Datenmengen
- Schnittstellen von Drittherstellern
 - PostGIS Shapefile Import / Export Manager
 - QGIS Plugins: DB-Verwaltung, DBManager, OGR/GDAL)
 - FME (Feature Manipulation Engine ([safe.com](https://www.safe.com)))

Import von Daten

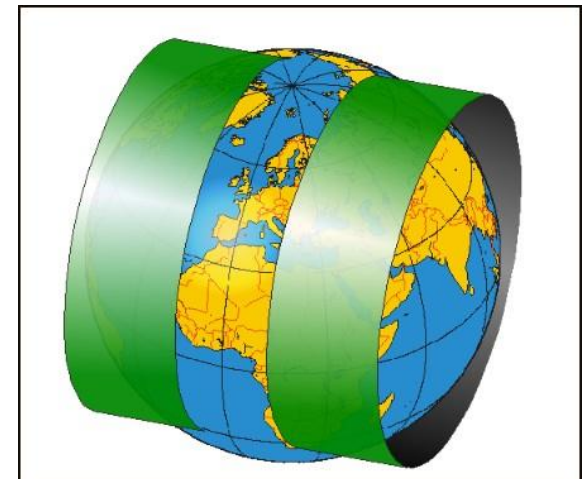
- Einladen (SHP→PG)
 - shp2pgsql → liefert alle möglichen Optionen
 - Umwandeln der Shapedaten in eine SQL Befehlsdatei mit dem Namen **wald.sql**
 - Shapedatei: **wald.shp**, GK Zone 4 (EPSG Code 31468), Tabellenname **wald**
 - shp2pgsql -s 31468 **wald.shp** **wald** >**wald.sql**
 - Ausführen des Skriptes, d.h. erzeugen der Tabelle **wald** in der Datenbank **thuringen**
 - psql -d **thuringen** -U user -f **wald.sql**
 - Realisierung mit einem Befehl (verketteten der o.g. Schritte)
 - shp2pgsql -s 31468 **wald.shp** **wald** | psql -d **thuringen** -U user

Export von Daten

- Erzeugen eines Shapes aus den Tabelleninformationen
 - Befehl im bin Verzeichnis von PostgreSQL:
pgsql2shp → liefert alle möglichen Optionen
 - `pgsql2shp -f wald.shp -u user thuringen wald`

Exkurs SRID (EPSG)

- Europäisches Terrestrisches Referenzsystem
 - 6 Grad breite vertikale Zonen, mittels transversaler Mercator Projektion verebnet
 - ETRS89/UTM Zone 32: 25832
 - ETRS89/UTM Zone 33: 25833
- WGS 84: 4326
- Gauss Krüger
 - Zone 2: 31466
 - Zone 3: 31467 (Thür)
 - Zone 4: 31468 (Thür)
 - Zone 5: 31469



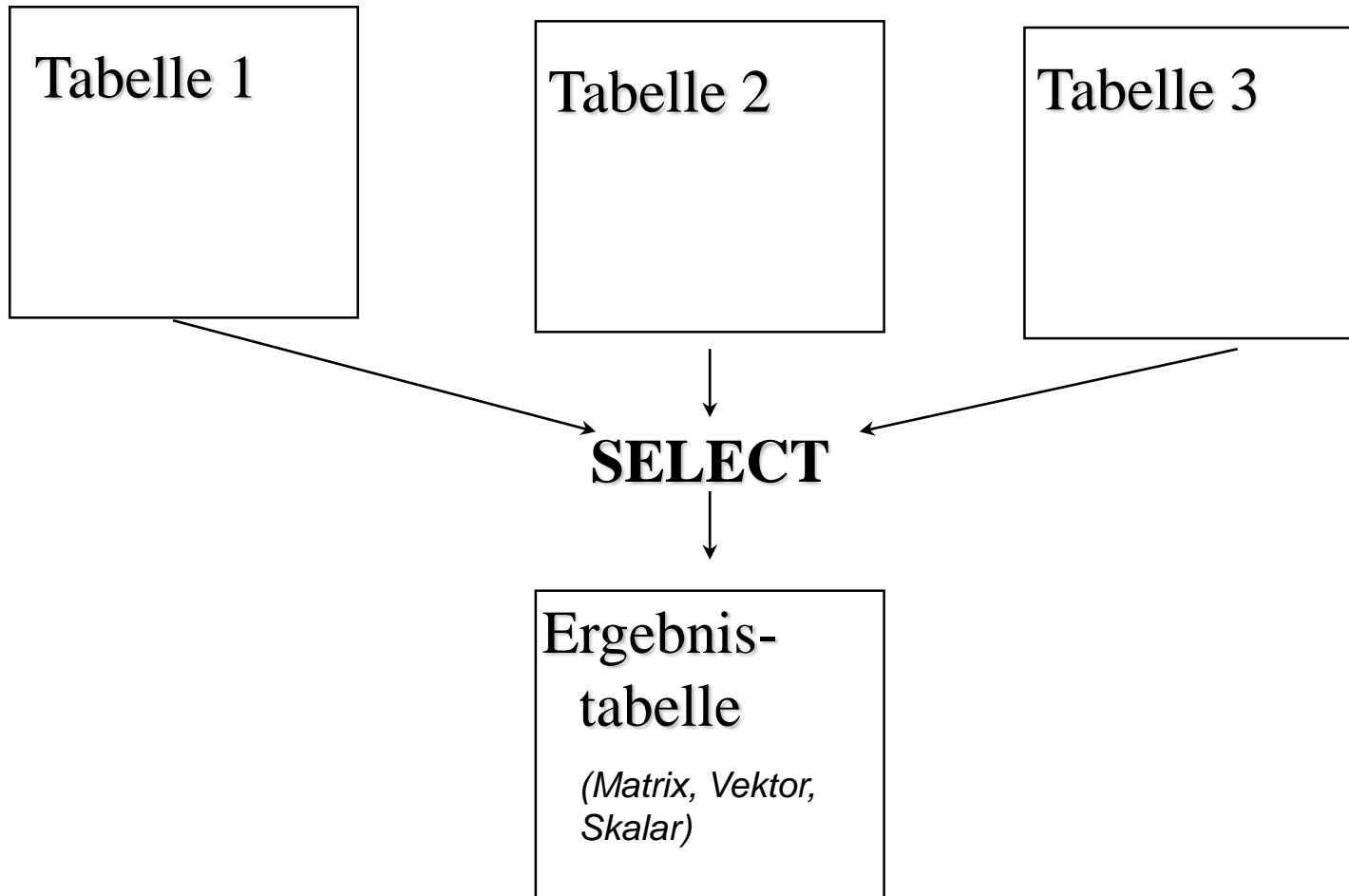
Ändern des Koordinatensystems bzw. Transformation der Daten

- Anzeige in der Tabelle (View) geometry_columns
- SRID falsch gesetzt (aber richtige Koordinaten in der Tabelle):
 - **ALTER TABLE** mytable **ALTER COLUMN** geom **TYPE** geometry(MultiPolygon,4326) **USING** ST_SetSRID(geom,25832);
- SRID falsch gesetzt (und die Koordinaten müssen transformiert werden):
 - **ALTER TABLE** mytable **ALTER COLUMN** geom **TYPE** geometry(MultiPolygon,25832) **USING** ST_Transform(geom,25832);

SELECT-Befehl

- wichtigster Befehl von SQL
- Datenabfrage
- Änderungsbefehle (INSERT, UPDATE, DELETE) nutzen SELECT-Befehl ebenfalls
- Auswahl bestimmter **Zeilen** und **Spalten** aus der Ursprungstabelle bzw. **Verknüpfung** von Tabellen

Tabellensicht beim SELECT-Befehl



Syntax und Reihenfolge

SELECT [**DISTINCT**] Auswahlliste =welche Spalten

FROM Quelle mit Verknüpfungen = woher kommen die Daten

[**WHERE** Where-Klausel = welche Zeilen]

[**GROUP BY** (Group-by-Attribut = Klassifizierung)+

[**HAVING** Having-Klausel = welche Klassen]]

[**ORDER BY** (Sortierungsattribut [**ASC**|**DESC**])+];



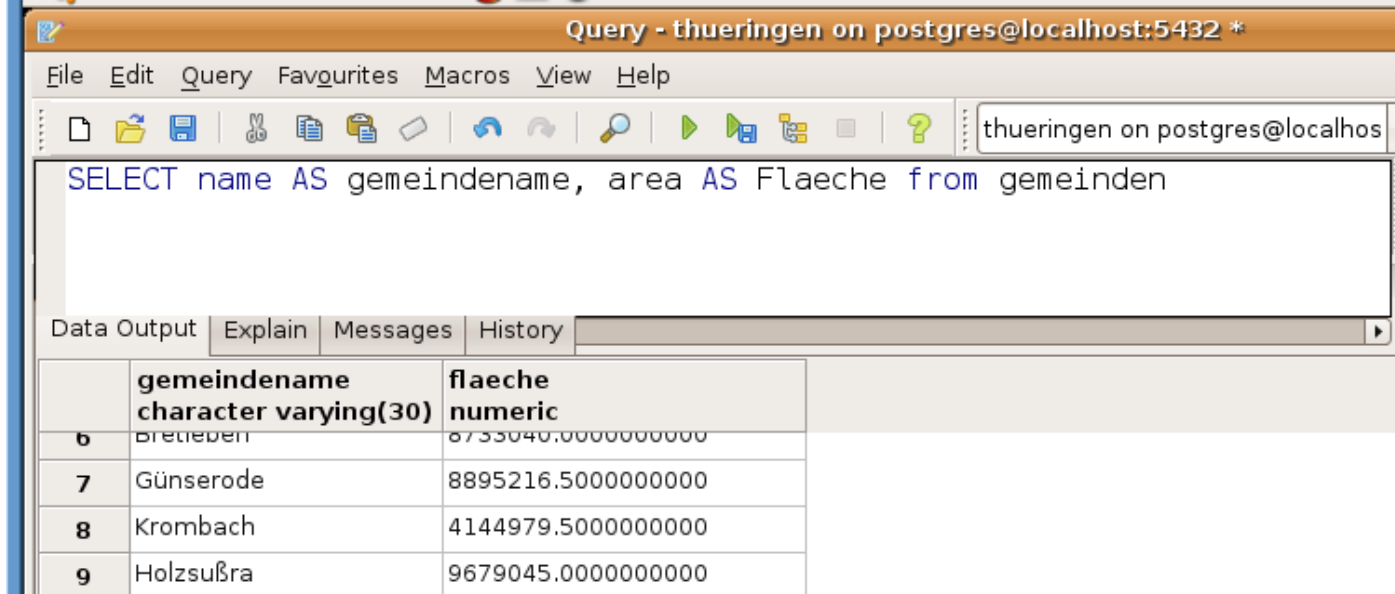
!! Die Reihenfolge der Schlüsselwörter ist wichtig !!

Spaltenauswahl

- Gib den Namen und die Fläche aller Gemeinden aus.
 - **SELECT** name, area **FROM** gemeinden
- Gib alle Informationen über die Gemeinden aus (d.h. alle Spalten und alle Zeilen).
 - **SELECT** * **FROM** gemeinden
 - * steht für **alle** Spalten der beteiligten Tabelle(**n**)

Umbenennen von Spalten mit **AS**


- Gib den Namen und die Fläche aller Gemeinden aus, weise der Ausgabe die Spaltennamen *Gemeindenname* und *Fläche* zu.
– **SELECT** name **AS** Gemeindenname, area **AS** Flaeche **FROM** gemeinden



The screenshot shows a PostgreSQL query editor window titled "Query - thuringen on postgres@localhost:5432 *". The query entered is: `SELECT name AS gemeindenname, area AS Flaeche from gemeinden`. The results are displayed in a table with two columns: **gemeindenname** (character varying(30)) and **flaeche** (numeric). The table contains four rows of data, with row numbers 6, 7, 8, and 9 visible in the first column.

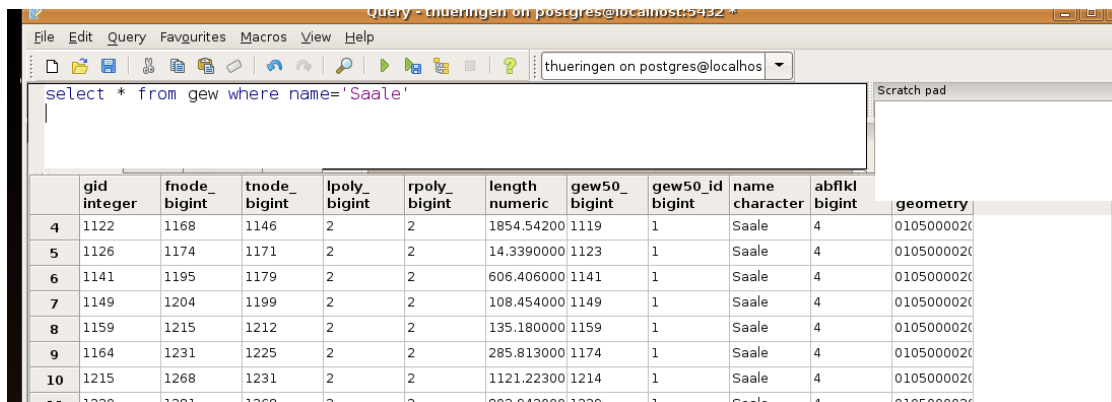
	gemeindenname character varying(30)	flaeche numeric
6	Bretleben	6733040.0000000000
7	Günserode	8895216.5000000000
8	Krombach	4144979.5000000000
9	Holzsußra	9679045.0000000000

Zeilenauswahl (WHERE-Klausel)

- Zeige alle Gemeinden mit einer Fläche größer als 100 km² 
(Exponentialschreibweise 100.000.000 = 100e6).
 - **SELECT** name **FROM** gemeinden
WHERE area > 100e6
- Welche Gemeindekennzahl hat Jena?
 - **SELECT** gkz **FROM** gemeinden
WHERE name='Jena'
 - Zeichenkettensuche mit Jokern → like, ilike und %:
 - **SELECT** gkz **FROM** gemeinden
WHERE name ilike '%Jena%'

DISTINCT

- Unterdrückt die mehrfache Ausgabe des Ergebnisses
- Beispiel: Tabelle gew enthält das digitalisierte Gewässernetz Thüringens



The screenshot shows a PostgreSQL query window titled "Query - thuringen on postgres@localhost:5432". The query entered is `select * from gew where name='Saale'`. The results are displayed in a table with 12 columns: gid, fnode_bigint, tnode_bigint, lpoly_bigint, rpoly_bigint, length_numeric, gew50_bigint, gew50_id_bigint, name_character, abflk_bigint, and geometry. The results show 10 rows of data for the 'Saale' river.

	gid integer	fnode_bigint	tnode_bigint	lpoly_bigint	rpoly_bigint	length_numeric	gew50_bigint	gew50_id_bigint	name_character	abflk_bigint	geometry
4	1122	1168	1146	2	2	1854.54200	1119	1	Saale	4	0105000020
5	1126	1174	1171	2	2	14.3390000	1123	1	Saale	4	0105000020
6	1141	1195	1179	2	2	606.406000	1141	1	Saale	4	0105000020
7	1149	1204	1199	2	2	108.454000	1149	1	Saale	4	0105000020
8	1159	1215	1212	2	2	135.180000	1159	1	Saale	4	0105000020
9	1164	1231	1225	2	2	285.813000	1174	1	Saale	4	0105000020
10	1215	1268	1231	2	2	1121.22300	1214	1	Saale	4	0105000020
11	1220	1281	1268	2	2	882.842000	1220	1	Saale	4	0105000020

Zeige alle digitalisierten Flüsse des Gewässernetzes

SELECT DISTINCT name **FROM** gew

Logische Operatoren

- AND, OR, NOT

- Zeige alle digitalisierten Gewässersegmente der Saale mit einer Länge größer einem Kilometer:

- **SELECT** * **FROM** gew **WHERE** name ilike '%saale%' **AND** length>1000



- Zeige alle digitalisierten Gewässersegmente der Saale und Wipper mit einer Länge größer einem Kilometer:

- **SELECT** * **FROM** gew **WHERE** (name ilike '%saale%' **OR** name ilike '%wipper%') **AND** length>1000



- Zeige alle Gewässersegmente mit bereits zugeordneter Benennung:


- **SELECT** * **FROM** gew **WHERE** name **IS NOT** NULL

- Prioritäten der Operatoren
- Klammerung

Sortierung von Ergebnissen

- ORDER BY [ASC|DESC]
 - Zeige alle Kreise aufsteigend geordnet nach ihrer Fläche.
 - SELECT name, area FROM kgr ORDER BY area ASC
 - SELECT name, area FROM kgr ORDER BY 2
 - Zeige alle Flußsegmente geordnet nach Namen (aufsteigend) und Länge (absteigend):
 - SELECT name, length FROM gew
ORDER BY name ASC, length DESC


Skalare Funktionen

- Funktion wird auf **jede Zeile der Tabelle** einzeln angewandt.
- Unterscheidung nach Argument und Funktionswert:
 - Zeichenkettenfunktionen: **LOWER**('Jena')='jena'
 - Mathematische Funktionen: **ROUND**(42.4382, 2)=42.44
 - Datumsfunktionen:
 - **EXTRACT**( DOT FROM TIMESTAMP '2016-02-16 20:38:40')=47
 - **NOW**()= '2017-09-25 01:30:40' (gegenwärtiger Zeitstempel)
 - Geometrische Funktionen: **ST_AREA**(the_geom) =
- Zeige alle Kreise und berechne deren Fläche:
 - **SELECT** kreis_name, **ST_AREA**(the_geom) **FROM** kgr


Aggregatfunktionen (1)

- Dienen der **gemeinsamen Verarbeitung von Tabellenzeilen** (Gruppenbildung für Zeilen)
- SQL-92 Standard definiert Aggregatfunktionen **MIN, MAX, SUM, AVG, COUNT**
- DB-Hersteller integrieren oft weitere Aggregatfunktionen, z.B. Finanzmathematik, räumliche Erweiterungen (**ST_UNION**)
- Beispiel: Was ist die größte Fläche eines Kreises in Thüringen?
 - **SELECT MAX(area) FROM kgr**
 - Ergebnis: 1307,..km²




Aggregatfunktionen (2)

- Zählen der Ergebniszeilen mit **COUNT**
 - Wieviel Kreise gibt es in Thüringen?
 - **SELECT COUNT(*) FROM** kgr
 - Ergebnis:23
- Unterdrückung von Mehrfachaufzählungen
 - Wie viel unterschiedliche Gewässer gibt es in Thüringen?
 - **SELECT COUNT(DISTINCT  name) FROM** gew
 - Ergebnis:850

Aggregatfunktionen (3)

- möglich:
 - Was ist die größte Fläche eines Kreises in Thüringen?
 - `SELECT MAX(area) FROM kgr` 
- nicht möglich:
 - Was ist die größte Fläche eines Kreises in Thüringen *und wie heißt dieser?*
 - `SELECT kreis_name, MAX(area) FROM kgr`
 - → Lösung mit Unterabfragen

Gruppierung von Ergebnissen(1)

- fasst Zeilen der Ergebnistabelle nach bestimmten Kriterien zusammen 
 - Wie lang sind die einzelnen Flüsse in Thüringen?
 - **SELECT** name, **SUM**(length) **FROM** gew 
GROUP BY name 

Gruppierung von Ergebnissen(2)

- Selektion der zusammengefassten Ergebnisse (Gruppen) durch HAVING
 - Welche Flüsse in Thüringen sind länger als 10 Kilometer?
 - **SELECT** name, **SUM**(length) **FROM** gew
GROUP BY name
HAVING **SUM**(length)>10000

Gruppierung von Ergebnissen(3)

- Was kann man anzeigen und was nicht und vor allem warum?
- **Falsch:**
 - **SELECT** s1,s2,s3 **FROM** tabelle **GROUP BY** s1,s2

s 1	s 2	s 3
a	x	2
b	y	4
b	y	6
a	x	10



s1	s2	s3
a	x	2,10
b	y	4,6

**!! Ein Wert
pro Zelle !!**

Gruppierung von Ergebnissen(4)

- Richtig:
 - `SELECT s1,s2, f(s3) tabelle GROUP BY s1,s2`
 - `SELECT s1, f(s2),s3 tabelle GROUP BY s1,s3`
 - **f** ist dabei ein Aggregatfunktion (z.B. MIN, MAX, SUM, COUNT)!

Komplexere Abfragen

Kombination verschiedener Tabellen

- Tabellenverknüpfungen (**Joins**)
- aufwendigsten und **teuersten** Operationen
- liefern Daten aus mehreren Tabellen
- Verknüpfungsarten:
 - Kartesisches Produkt,
 - Gleichheitsverbindung (Equijoin),
 - Äußerer Verbund (Outer Join)

Kreuzprodukt oder kartesisches Produkt

(crossjoin), Grundlage für geometrische Abfragen - beliebige Kombination aller Zeilen

Personen

P_NR	Name	Vorname
P1	Moldenhauer	Steffen
P2	Löffler	Ralf

Hobbys

P_NR	Hobbys
P1	Volleyball
P2	Radsport

SELECT * FROM personen CROSS JOIN hobbies

P_NR	Name	Vorname	P_NR	Hobbys
P1	Moldenhauer	Steffen	P1	Volleyball
P1	Moldenhauer	Steffen	P2	Radsport
P2	Löffler	Ralf	P1	Volleyball
P2	Löffler	Ralf	P2	Radsport

Gleichheitsverbund



(Equijoin), Kombination der Zeilen bei Gleichheit eines oder mehrerer Felder

Personen

P_NR	Name	Vorname
P1	Moldenhauer	Steffen
P2	Löffler	Ralf

Hobbys

P_NR	Hobbys
P1	Volleyball
P2	Radsport

„Gib alle Daten der Personen und ihrer Hobbys aus“

```
SELECT * FROM personen INNER JOIN hobbies ON  
personen.p_nr=hobbies.p_nr
```

P_NR	Name	Vorname	P_NR	Hobbys
P1	Moldenhauer	Steffen	P1	Volleyball
P2	Löffler	Ralf	P2	Radsport

Gleichheitsverbund

(Equijoin)

Personen

P_NR	Name	Vorname
P1	Moldenhauer	Steffen
P2	Löffler	Ralf
P3	Kaiser	Axel

Hobbys

P_NR	Hobbys
P1	Volleyball
P2	Radsport
P2	Fussball

„Gib alle Daten der Personen und ihrer Hobbys aus“



```
SELECT personen.*,hobbys.hobbys  
FROM personen INNER JOIN hobbys ON personen.p_nr=hobbys.p_nr
```

P_NR	Name	Vorname	Hobbys
P1	Moldenhauer	Steffen	Volleyball
P2	Löffler	Ralf	Radsport
P2	Löffler	Ralf	Fussball

Was ist mit Axel Kaiser?

Äußerer Verbund

(outer join), betrachtet auch Zeilen ohne Entsprechung in den beteiligten Tabellen

Personen

P_NR	Name	Vorname
P1	Moldenhauer	Steffen
P2	Löffler	Ralf
P3	Kaiser	Axel



Hobbys

P_NR	Hobbys
P1	Volleyball
P2	Radsport
P2	Fussball

„Gib alle Daten der Personen und ihrer Hobbys aus, (auch wenn sie keine Hobbys besitzen !!)“

SELECT personen.*,hobbys.hobbys **FROM**
personen **LEFT OUTER JOIN** hobbys **ON** personen.p_nr=hobbys.p_nr

P_NR	Name	Vorname	Hobbys
P1	Moldenhauer	Steffen	Volleyball
P2	Löffler	Ralf	Radsport
P2	Löffler	Ralf	Fussball
P3	Kaiser	Axel	null

Gleichheitsverbund mit drei Tabellen

(Equijoin)

Personen

P_NR	Name	Vorname
P1	Moldenhauer	Steffen
P2	Löffler	Ralf
P3	Kaiser	Axel

 arbeitet_in

P_NR	A_NR
P1	A3
P2	A1
P3	A2

Abteilungen

A_NR	Name
A1	Programmierung
A2	Fernerkundung
A3	GIS

„Gib die Namen der Personen und ihrer zugeordneten Abteilungen aus.“

SELECT

vorname, personen.name AS nachname, abteilungen.name AS abteilung
FROM personen INNER JOIN arbeitet_in ON personen.p_nr=arbeitet_in.p_nr
INNER JOIN abteilungen ON arbeitet_in.a_nr = abteilungen.a_nr

Vorname	Nachname	Abteilung
Steffen	Moldenhauer	GIS
Ralf	Löffler	Programmierung
Axel	Kaiser	Fernerkundung

Unterabfragen (Subqueries)

- Was ist die größte Fläche eines Kreises in Thüringen *und wie heißt dieser?*

1.Versuch:

- **SELECT** kreis_name, **MAX**(area) **FROM** kgr

2.Versuch

- **SELECT** kreis_name, area **FROM** kgr **WHERE**
area=**MAX**

Einzeilige Unterabfragen

- Unterabfrage darf nur eine Zeile als Ergebnis zurückliefern
- Verwendung nur mit WHERE und HAVING
- Lösung des vorherigen Problems:
`SELECT kreis_name, area FROM kgr
WHERE area =
(SELECT MAX(area) FROM kgr)`

Mehrzeilige Unterabfragen

- Unterabfragen liefern mehr als eine Zeile zurück
 - Verarbeitung mit dem **IN** Operator
 - Verarbeitung der Ergebnisse mit dem **ALL** Operator
 - Vergleicht **jeden** Wert der Unterabfrage mit dem äußeren Element.
 - Benutzung immer zusammen mit den Operatoren =,!=,<,<=,>,>=

Beispiel IN Operator

- Anfrage:
 - Gibt es in Thüringen gleiche Gemeinde- und Flussnamen?

```
SELECT gemeinden.name FROM gemeinden
where gemeinden.name IN
    (SELECT gew.name FROM gew)
```

Beispiel ALL Operator

„Schachtelung von Aggregatfunktionen“

- Anfrage:
 - Was ist der längste Fluss in Thüringen?
 - **SELECT** name, **SUM**(length) **FROM** gew
GROUP BY name
HAVING SUM(length) **>=** **ALL**
(**SELECT SUM**(length) **FROM** gew **GROUP BY** name)

Beispiel ALL Operator

„Schachtelung von Aggregatfunktionen“

- Anfrage:
 - Was ist der längste (benamte) Fluss in Thüringen?
 - **SELECT** name, **SUM**(length) **FROM** gew **WHERE** name IS NOT NULL **GROUP BY** name **HAVING SUM**(length) >= **ALL**
(**SELECT SUM**(length) **FROM** gew **WHERE** name IS NOT NULL **GROUP BY** name)

Benutzersichten (VIEWS)

Personal_Empfang

Name Vorname Abteilung Funktion Telefonnummer

Personal_Lohn

Name Vorname Gehalt

Personal

Name	Vorname	Abteilung	Funktion	Telefonnummer	Gehalt
Löffler	Ralf	Software	Java Entwickler	03641/3038-12	3450
Moldenhauer	Steffen	Software	Analyst	03641/3038-16	4500
Kaiser	Axel	GIS	A-Entwickler	03641/3038-53	2500
Köhler	Steffen	GIS	GeoDB-Entwickler	03641/3038-30	3000

Benutzersichten (VIEWS)

- Reale Tabellen (**CREATE TABLE ...**)
- virtuelle Tabellen (**CREATE VIEW ...**)
 - werden wie Tabellen behandelt
 - basieren auf Tabellen und anderen Sichten, d.h. Inhalt wird jeweils neu extrahiert.
 - Vergleichbar mit einem Tabellenfilter für Spalten und Zeilen
 - spezielle Darstellung des Datenbankinhaltes
 - Realisierung durch Kombination mit **SELECT** Anweisung

Vorteile von Views

- Sicherheit vor unerlaubten Datenzugriff
- Vermeidung von Inkonsistenzen
- Nutzerdefinierte Darstellung möglich
- Verbergen der Struktur einer Datenbank
- Entspricht einem „Abfragelayer“ in ArcGIS
- Können durch GI-Systeme (z.B. ArcGIS, QGIS) als **Layer** verarbeitet werden.

Erstellen von Views

- **CREATE VIEW** viewname **AS SELECT** ...
- Erzeugen einer (virtuellen) Kreistabelle, welche nur Kreise mit einer Fläche $>50 \text{ km}^2$ enthält.
 - **CREATE VIEW** kgr50km2 **AS**
SELECT * FROM kgr **WHERE** area $>50\text{e}6$
 - Falls syntaktisch alles korrekt, entsteht eine neue virtuelle Tabelle, welche jetzt mit *select * from kgr50km2* aufgerufen werden kann.
 - Für eine Änderung muss die View gelöscht und wieder neu erzeugt werden. Befehl: *drop view kgr50km2*
 - Jetzt kann die Änderung mittels erneuter Erzeugung durchgeführt werden: *create view as ...*
 - *Alternative: CREATE OR REPLACE VIEW ...*

Voraussetzungen für die Nutzung PostgreSQL (PostGIS) und QuantumGIS

- Die folgenden beiden Spalten sind minimal für die Darstellung notwendig:
 1. Spalte mit einer ID (*Primärschlüssel*)
 - Beim Datenimport einer Shape Datei wird die Spalte **gid** (Datentyp Integer/Serial) erzeugt.
 2. Spalte mit der Geometrie (*Datentyp GEOMETRY* oder davon abgeleitet (*POINT, LINE, POLYGON, ...*)
 - Standardmäßig wird die Spalte beim Datenimport im PostGIS **the_geom** (seit Version 2.0 **geom**) benannt. Diese enthält die Informationen der Shapedatei *shp*.