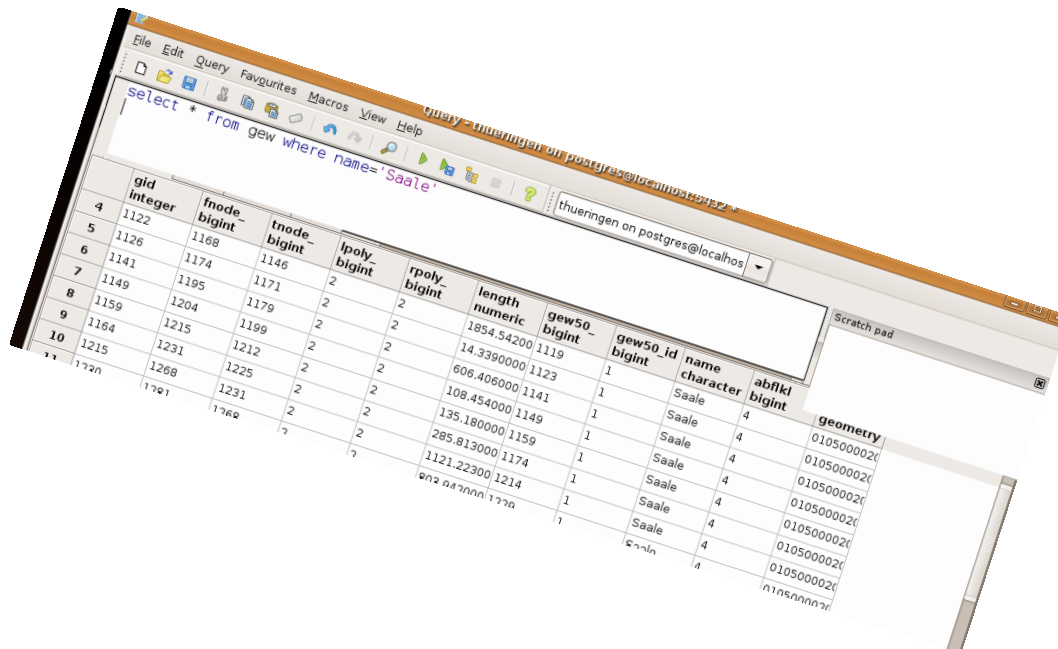


GIS Kurs

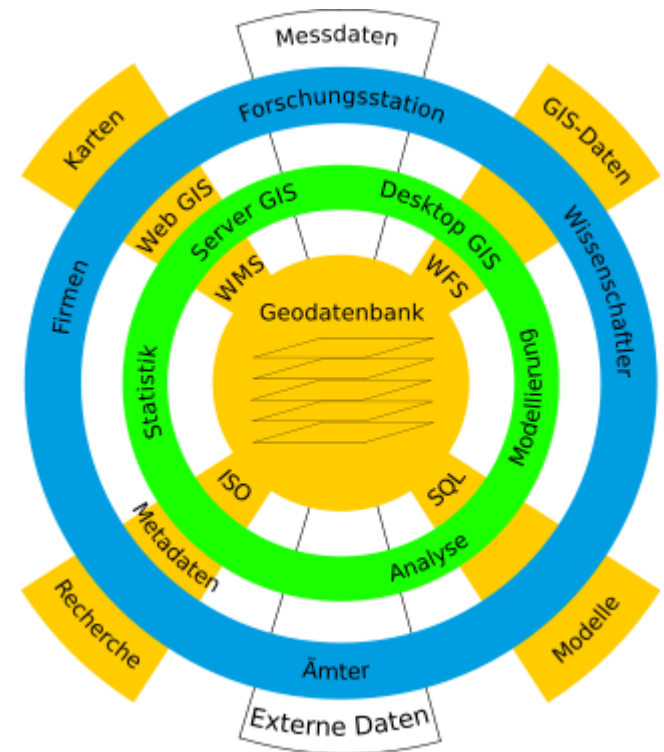
- Geodatenbanken -

Carsten Busch



Query: `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	bigint	geometry
4	1122	1168	1171	2	2	1854.54200	1119	1	1	Saale	4	4	0105000020	
5	1126	1174	1171	2	2	14.3390000	1123	1	1	Saale	4	4	0105000020	
6	1141	1195	1179	2	2	606.406000	1141	1	1	Saale	4	4	0105000020	
7	1149	1204	1199	2	2	108.454000	1149	1	1	Saale	4	4	0105000020	
8	1159	1215	1212	2	2	135.180000	1159	1	1	Saale	4	4	0105000020	
9	1164	1231	1225	2	2	285.813000	1174	1	1	Saale	4	4	0105000020	
10	1215	1268	1231	2	2	1121.22300	1214	1	1	Saale	4	4	0105000020	
11	1220	1268	1231	2	2	1121.22300	1214	1	1	Saale	4	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

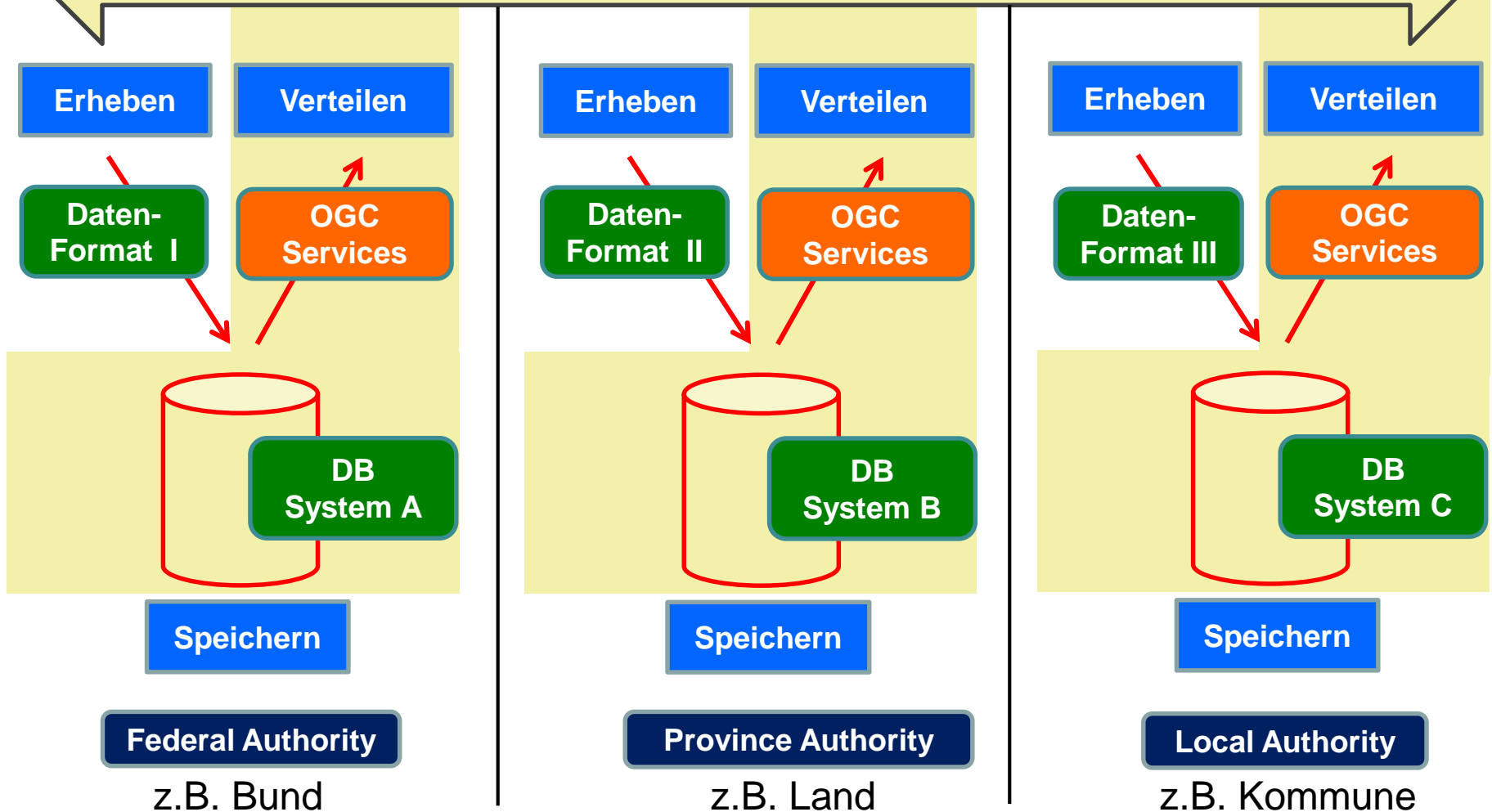
[Geodatenbanken]

- **Trends**

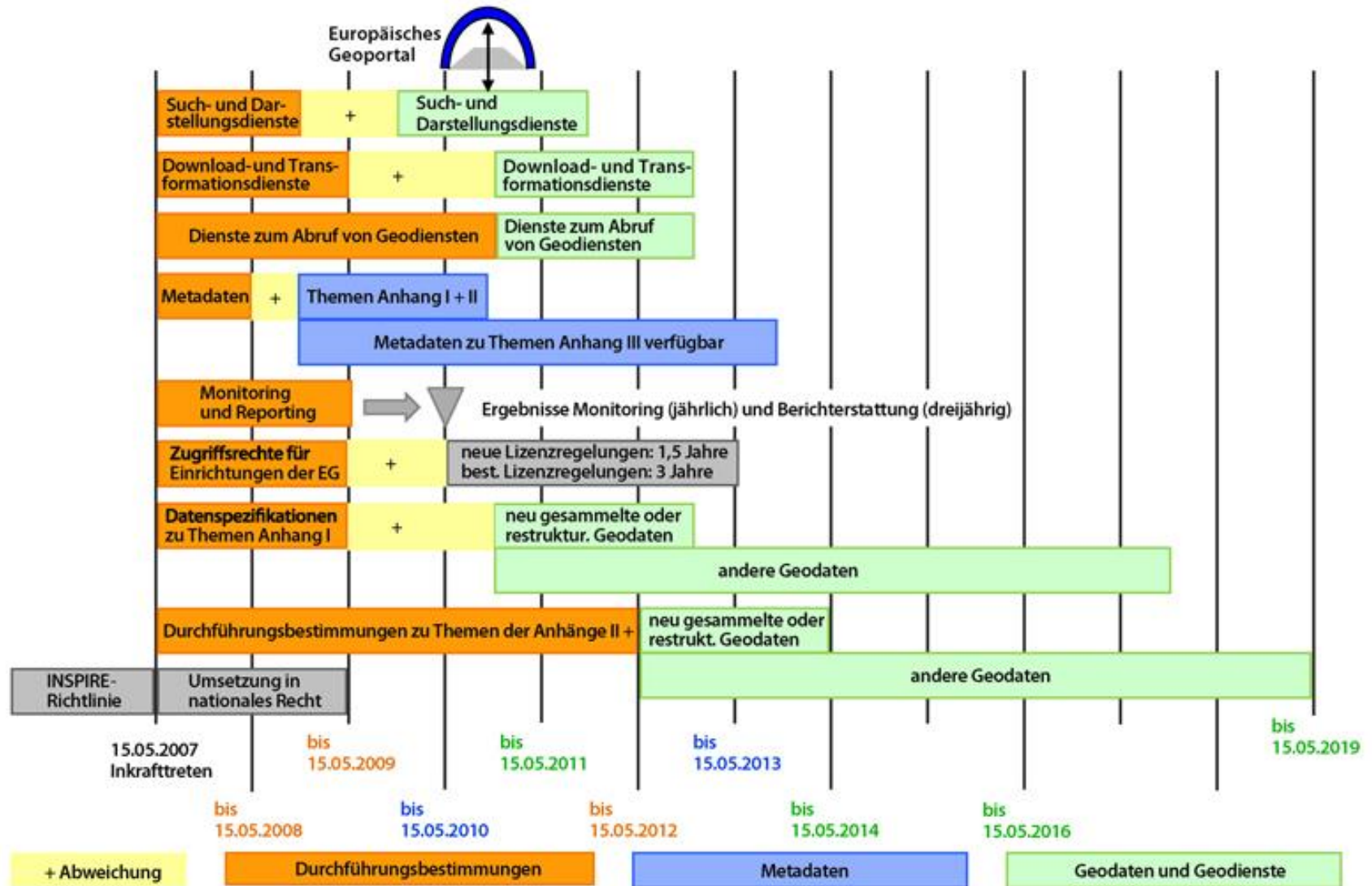
- Gruppenorientierte Entwicklungsarbeit, **gleichzeitige** Bearbeitung einer Zeichnung/GIS-Daten durch **mehrere** Anwender
- Bearbeitung einer Zeichnung/GIS-Daten an **unterschiedlichen** Standorten
- Nutzung des WWW → WebGIS
- Umsetzung nationaler/internationaler Standards
 - **GDI** Geodateninfrastruktur
 - **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

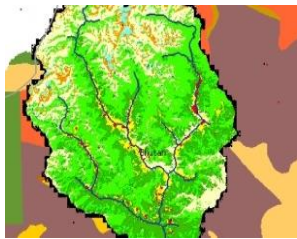


Databases

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



Spatial SQL, SOS
(Sensor Observation Service)

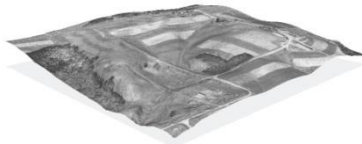


Maps

(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

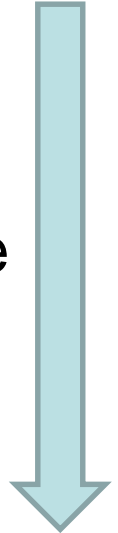
(e.g. street information as Shape, DXF, SVG)



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

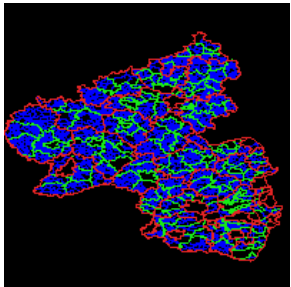


→ Zentralisierung der Geodaten (GeoDB), Zugriff über webbasierter GIS Dienste ←

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

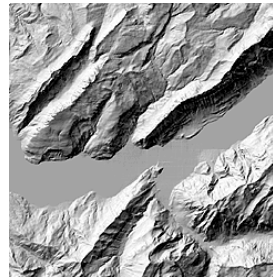
Woher
kommen die
Daten ?



VG Rheinland Pfalz



DLM 250



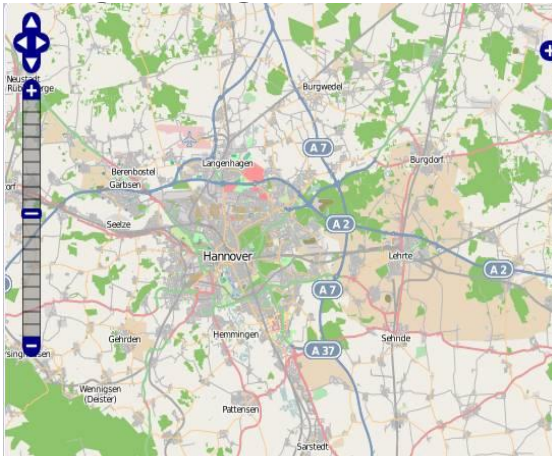
DHM25



TK25

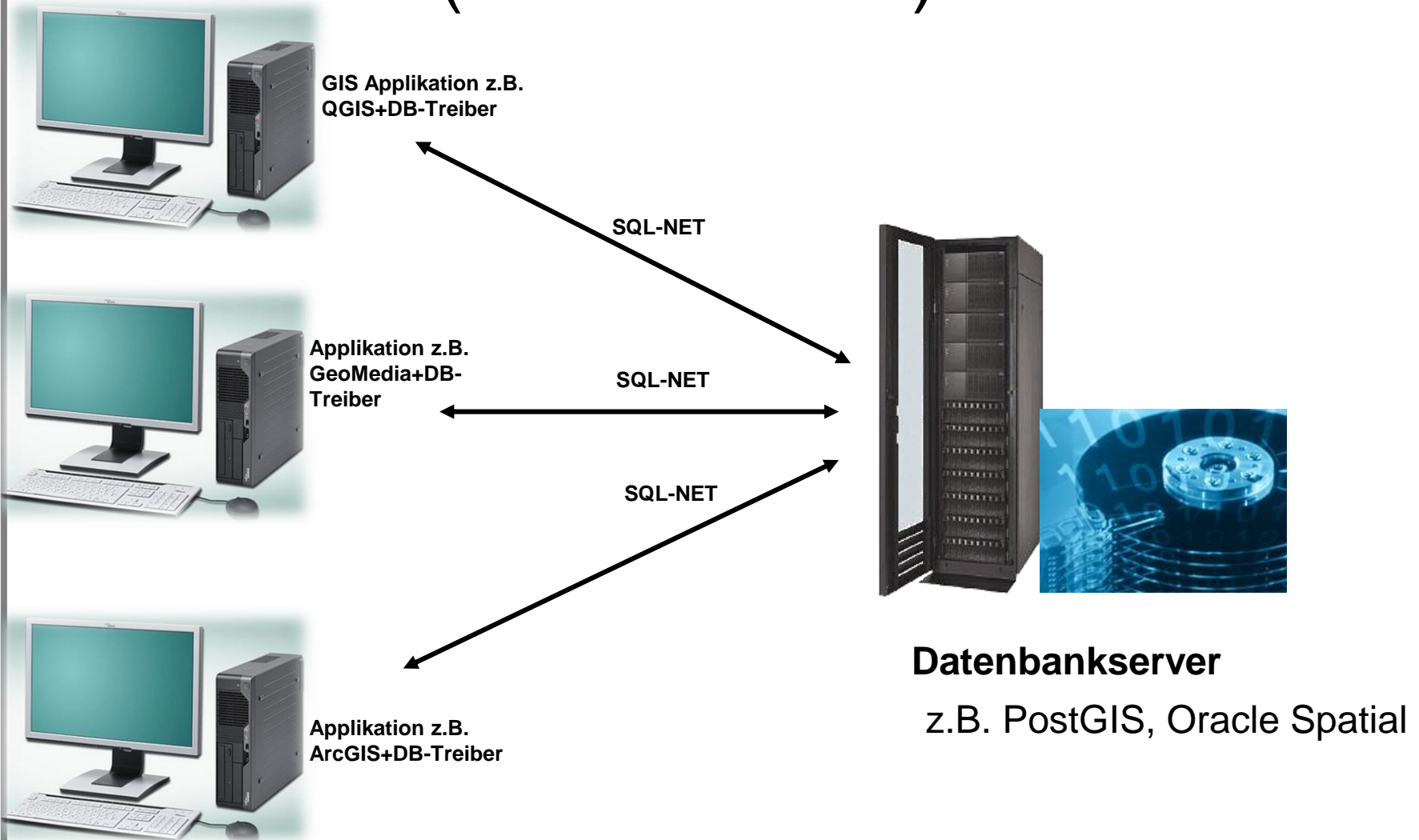
Datenanbieter (ii)

- 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

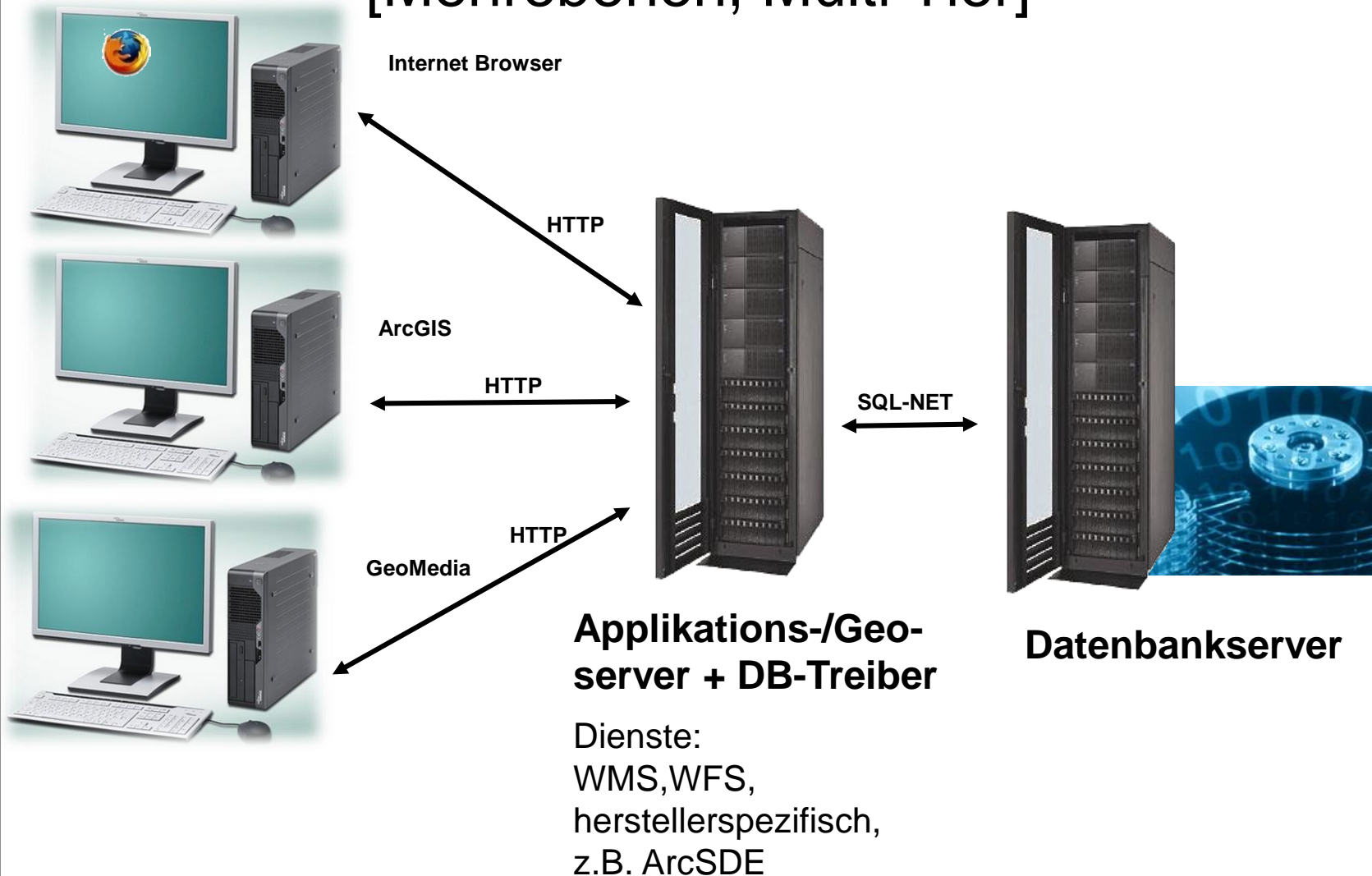


Open Street Map

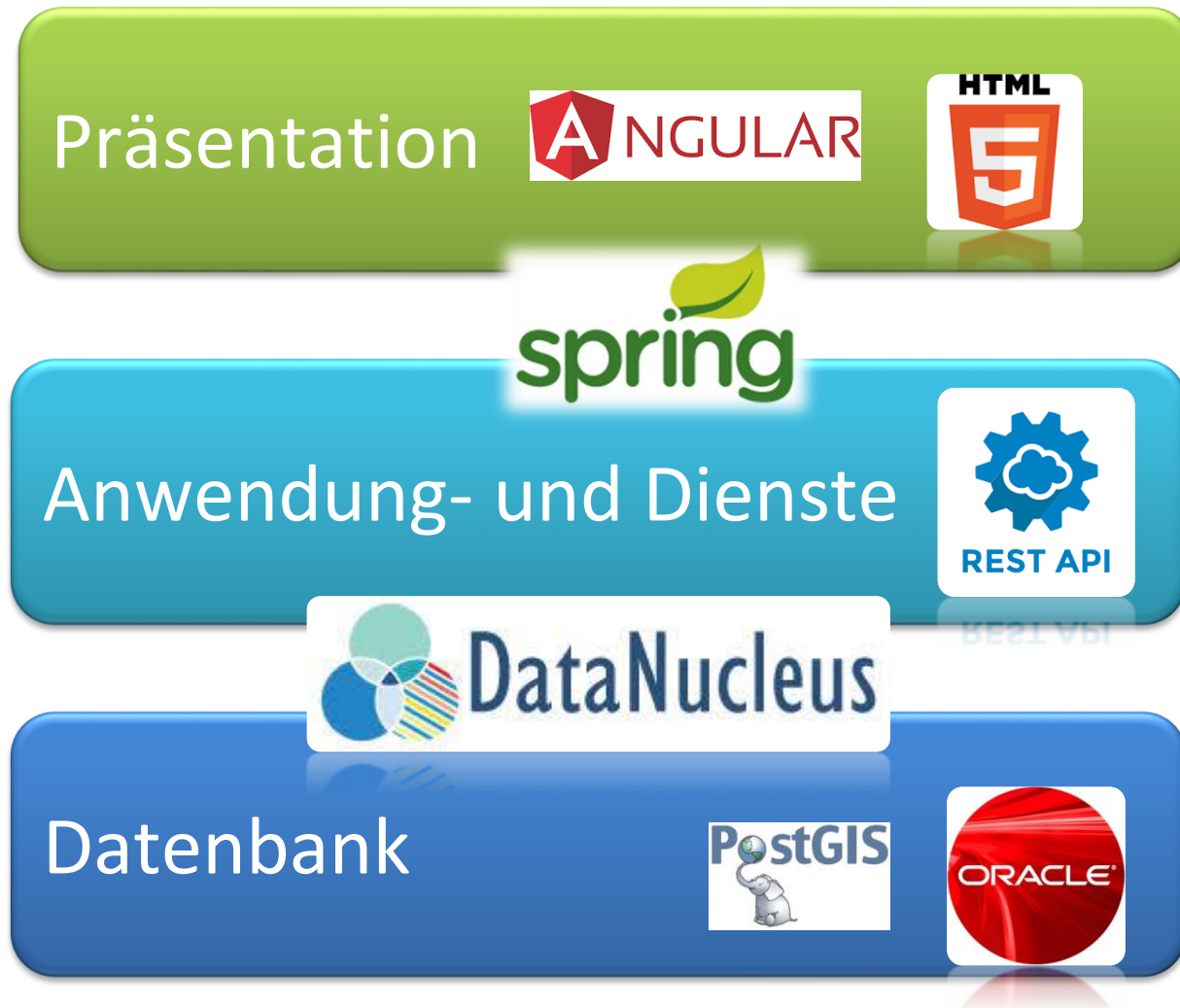
Zwei Ebenen Architektur (Client-Server)



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.

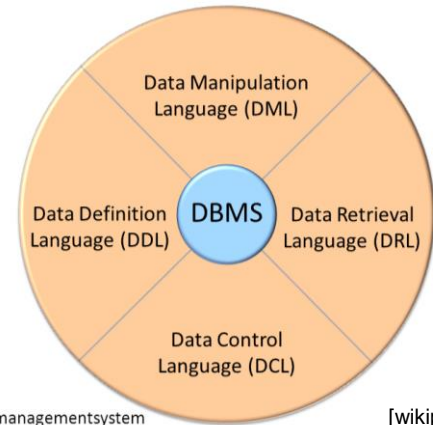
SQL

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.)



[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)
```

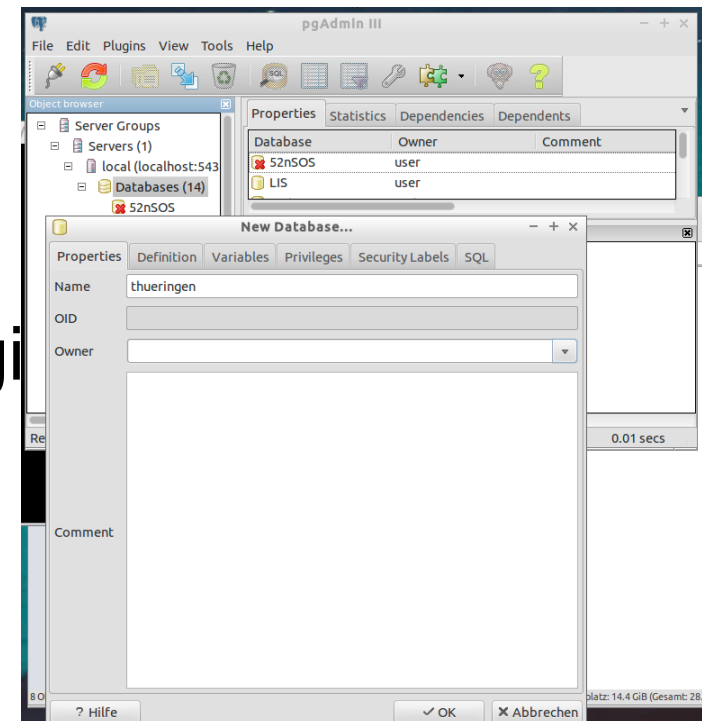
[*Was*, mengenmäßige Beschreibung]

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

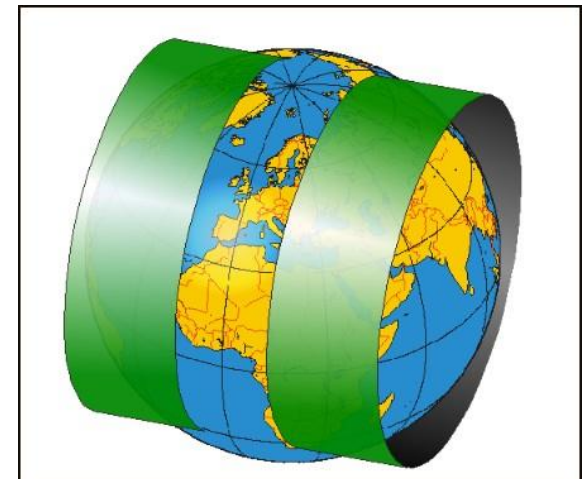
- Einladen (SHP→PG)
 - shp2pgsql → liefert alle möglichen Optionen
 - Umwandeln der Shapedaten in eine SQL Befehlsdatei mit dem Namen **wald.sql**
 - Shapedatei: **wald.shp**, GK Zone 4 (EPSG Code 31468), Tabellenname **wald**
 - shp2pgsql -s 31468 **wald.shp** **wald** >**wald.sql**
 - Ausführen des 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



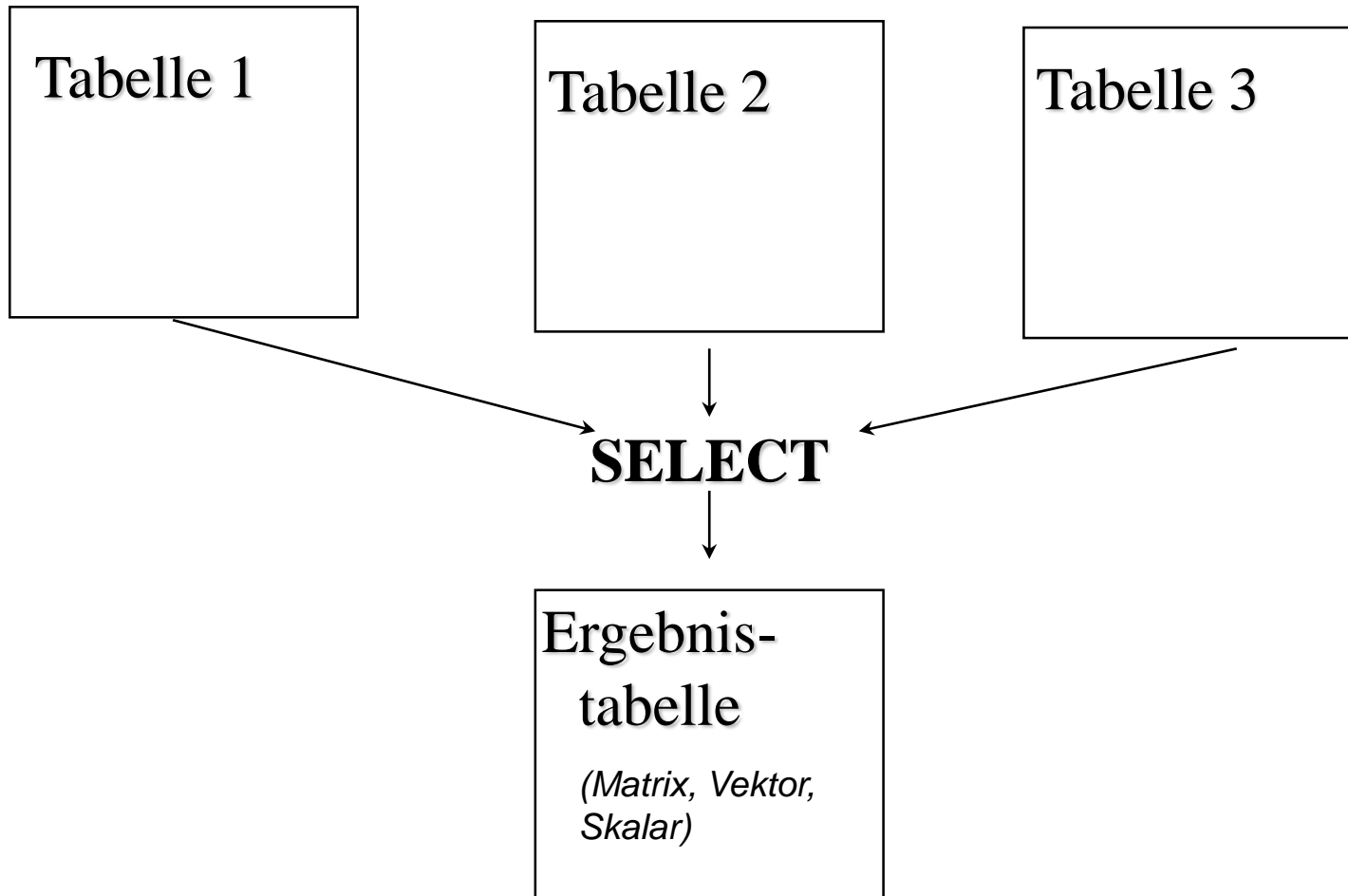
Ändern des Koordinatensystems bzw. Transformation der Daten

- 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**])+];

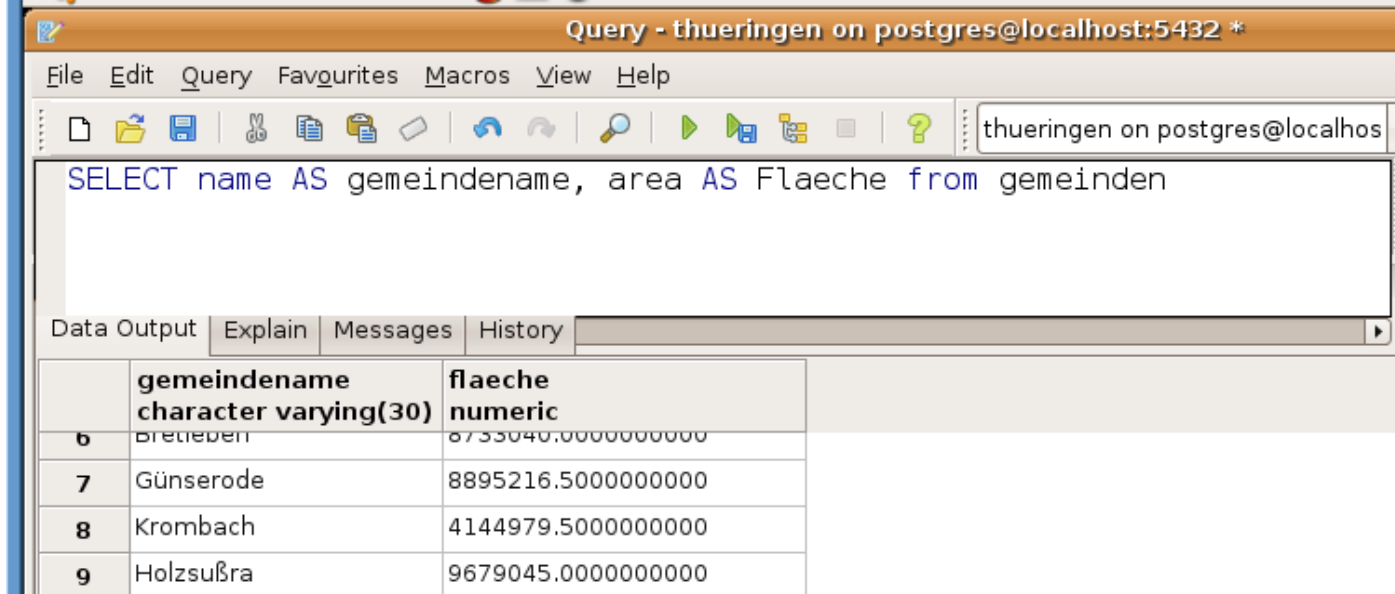
!! Die Reihenfolge der Schlüsselwörter ist wichtig !!

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.

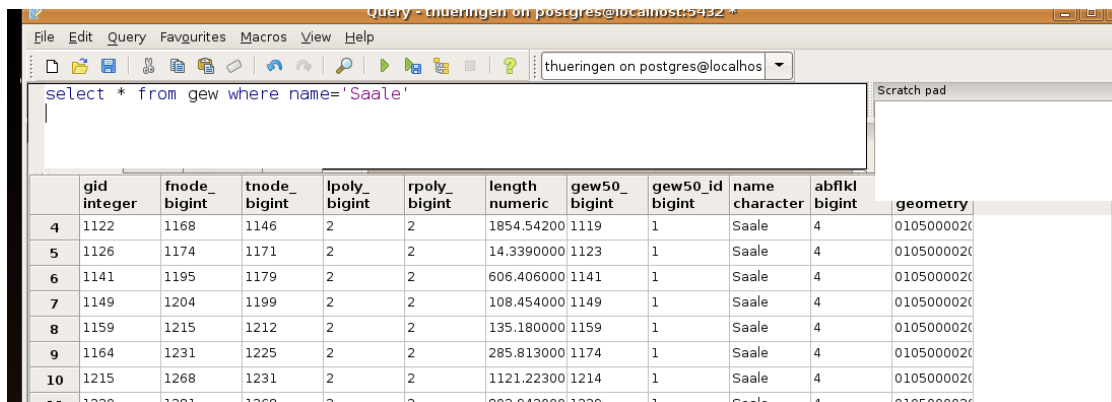
	gemeindenname character varying(30)	flaeche numeric
6	Bretleben	6733040.0000000000
7	Günserode	8895216.5000000000
8	Krombach	4144979.5000000000
9	Holzsußra	9679045.0000000000

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 table contains 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**(DOY 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 km² enthält.
 - **CREATE VIEW** kgr50km2 **AS**
SELECT * FROM kgr **WHERE** area>50e6
 - 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*.

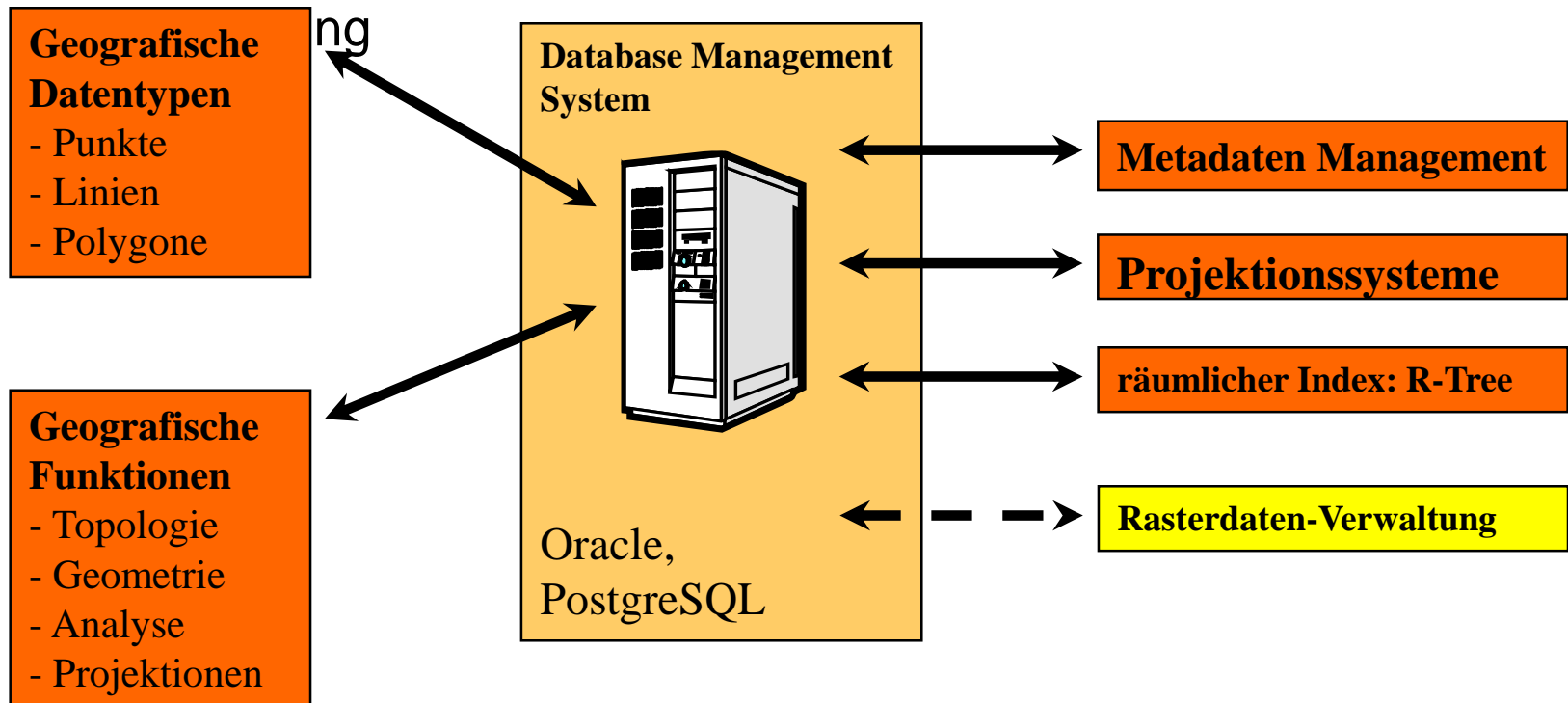
Verwaltung von GIS Daten in Datenbanken

-relevante Produkte-

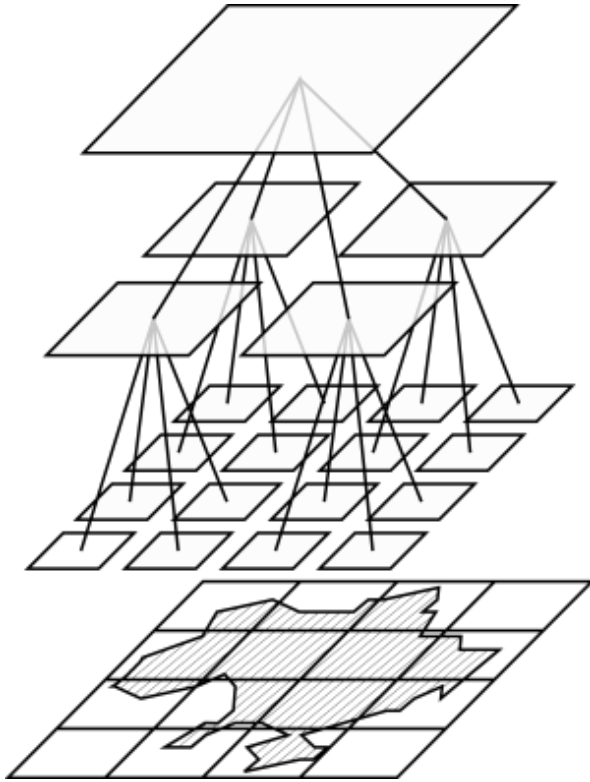
- Oracle, Geodatenserver
 - Oracle Locator (Verwaltung von Vektoren)
 - Oracle Spatial (Verwaltung von Rasterdaten)
- ESRI (ARC View/ARC Info)
 - Spatial Database Engine (SDE)
- PostgreSQL
 - Erweiterung PostGIS

Geodatenbanken

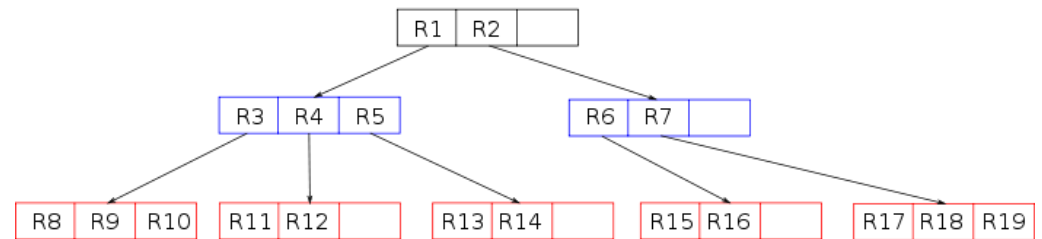
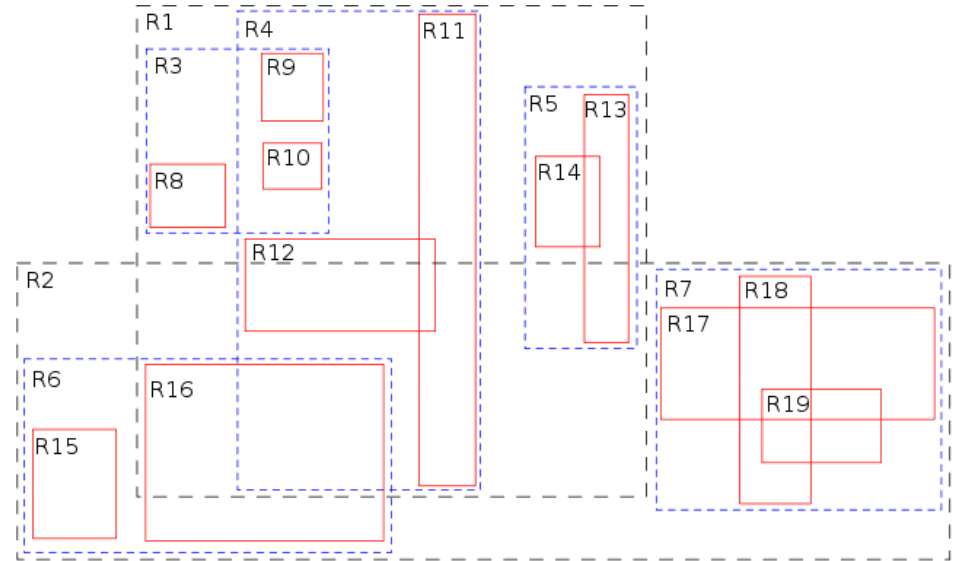
- Relationales Modell kennt wenige Datentypen: Zahlen, Zeichenketten, Wahrheitswerte (Boolean), Datum
- Geo-(objekt)-relationales Modell fügt weitere Datentypen mit entsprechenden Funktionen hinzu
 - Datentypen mit Raumbezug und Funktionen zur geometrischen



Räumliche Indizes




Quad -Trees

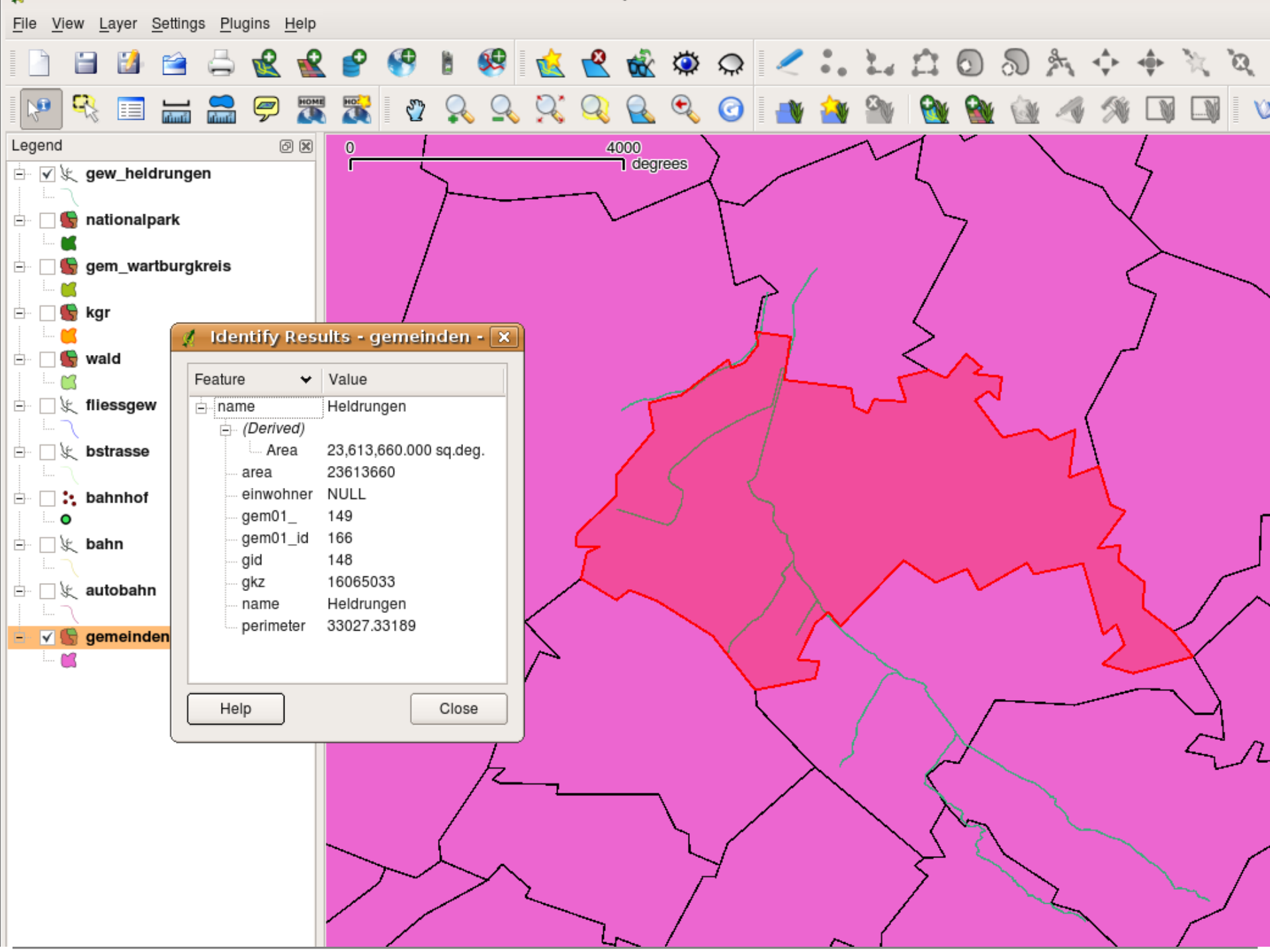


R-Trees

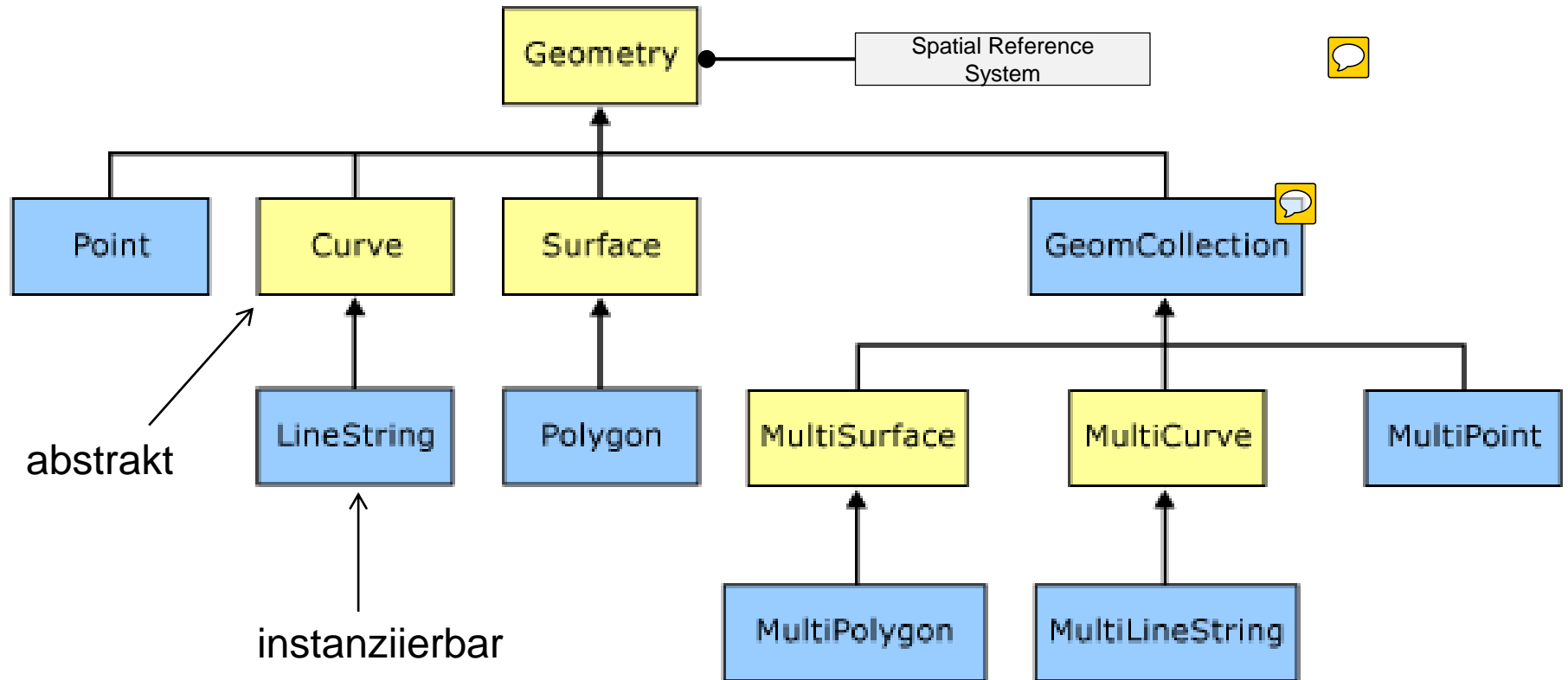
Prinzip

- Standard-SQL und räumliche SQL-Befehle können frei gemischt werden.
- Beispiel: Welche Gewässer gibt es in der Gemeinde Heldringen?

```
SELECT DISTINCT gew.name AS Gewaesser  
FROM gemeinden CROSS JOIN  gew  
WHERE  
ST_INTERSECTS(gemeinden.the_geom, gew.the_geom)  
AND  
gemeinden.name ilike '%Heldringen%'
```

Simple Feature for SQL (SFS) Modell



[SQL Geometry
Type hierarchy,
opengeospatial.org]

Allgemeiner Basistyp GEOMETRY

Generischer Typ für
geografische Objekte

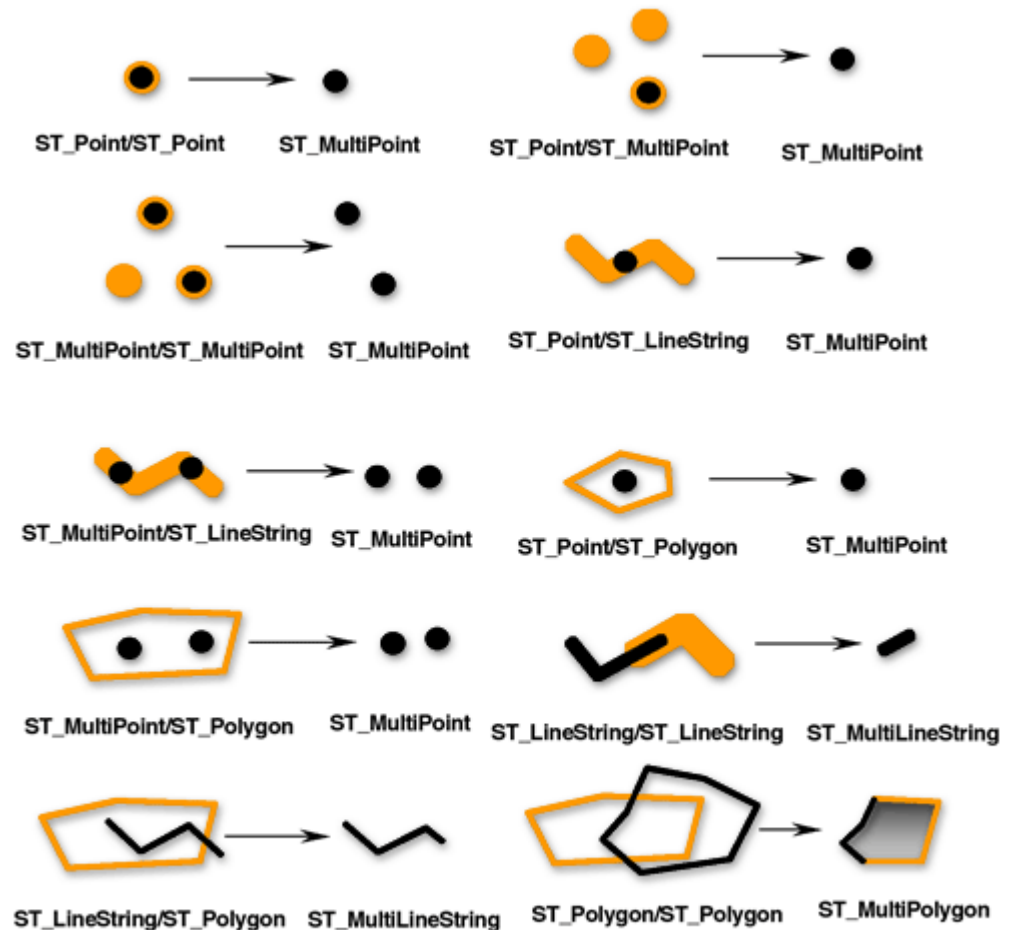
Ist der Basistyp(-klasse) für
alle anderen geometrischen
Typen

→ Polymorphie

Bsp. Funktion

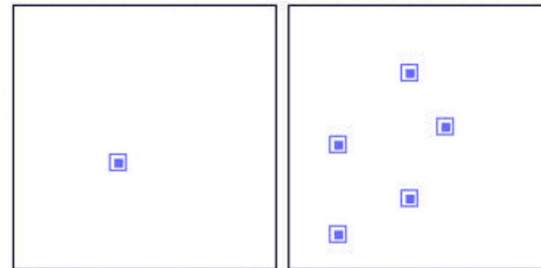
st_intersection 

(Verschneidung zweier
beliebiger Geometrien)
liefert Geometry, das
konkrete Ergebnis kann
vom Typ point, line oder
polygon sein



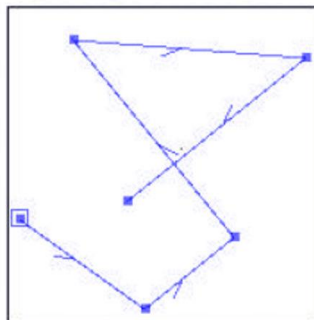
[esri.com]

PostGIS Types, abgeleitet von Geometry

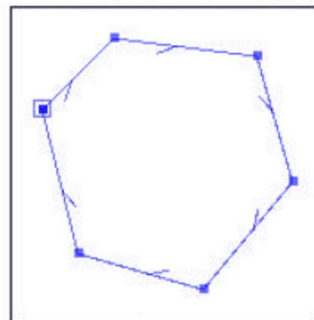


Point

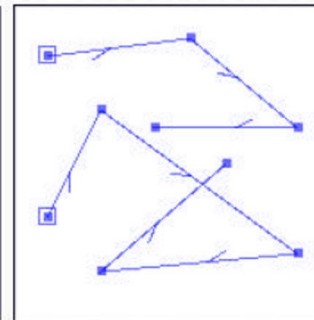
MultiPoint



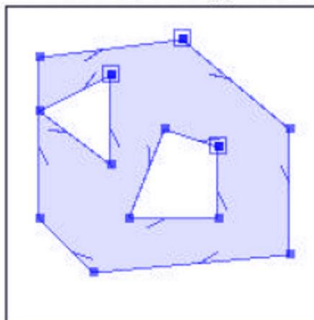
LineString



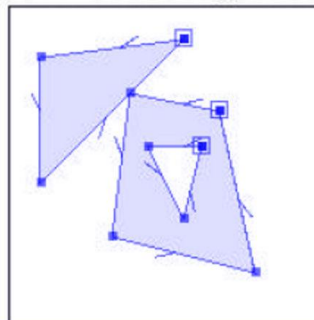
LinearRing



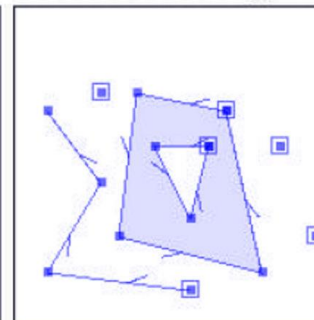
MultiLineString



Polygon




MultiPolygon



GeometryCollection

[postgis.net]

Darstellung einer Geometrie als Zeichenkette

- Bezeichnung: **WKT** (well known text) representation
- Darstellung des Types (point, linestring, polygon...) und der zugehörigen Koordinaten als Zeichenkette
- PostGIS Funktion: **ST_ASTEXT**(geom) 
- **Achtung:** Bei großen Geometrien ist die **WKT** Darstellung wesentlich größer als Binärdarstellung!

Beispiele WKT

Geometry Type	SQL Text Literal Representation	Comment
Point	<code>'POINT (10 10)'</code>	a Point
LineString	<code>'LINESTRING (10 10, 20 20, 30 40)'</code>	a LineString with 3 points
Polygon	<code>'POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))'</code>	a Polygon with 1 exterior ring and 0 interior rings
Multipoint	<code>'MULTIPOINT (10 10, 20 20)'</code>	a MultiPoint with 2 point
MultiLineString	<code>'MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))'</code>	a MultiLineString with 2 linestrings
MultiPolygon	<code>'MULTIPOLYGON (((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60)))'</code>	a MultiPolygon with 2 polygons
GeomCollection	<code>'GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))'</code>	a GeometryCollection consisting of 2 Point values and a LineString value

Definition geometrischer Funktionen auf Basis des Dimensionally Extended 9 Intersection Model (DE-9IM),

[Egenhofer 1990, Clementini 1993]

-PostGIS Funktion ST_Relate-



Beispiel:
Verschneidung
zweier
überlappender
Polygone

Ergebnis-Dimension	Geom. Objekt
2	Polygon
1	Linie
0	Punkt
-1	Keine Schnittmenge



	<i>Interior</i>	<i>Boundary</i>	<i>Exterior</i>
<i>Interior</i>	 $\dim(\dots) = 2$	 $\dim(\dots) = 1$	 $\dim(\dots) = 2$
<i>Boundary</i>	 $\dim(\dots) = 1$	 $\dim(\dots) = 0$	 $\dim(\dots) = 1$
<i>Exterior</i>	 $\dim(\dots) = 2$	 $\dim(\dots) = 1$	 $\dim(\dots) = 2$

[Postgis.org]

PostGIS Funktionen


Rückgabewert: Funktionsname (Argument)

- Eine Geometrie als Argument
 - Float : **ST_PERIMETER**(Geometry)
 - Float : **ST_AREA**(Geometry)
 - Geometry : **ST_CENTROID**(Geometry)
 - Geometry : **ST_BUFFER**(Geometry,Float)
 - Geometry : **ST_BUFFER**(Geometry,Float)
 - Geometry : **ST_TRANSFORM**(Geometry,Integer)
 - Text : **ST_ASTEXT**(Geometry)
 - Float: **ST_X**(Geometry)
 - Float: **ST_Y**(Geometry)

PostGIS Funktionen

- Mehrere Geometrien als Argument
 - Float : **ST_DISTANCE**(Geometry,Geometry)
 - Boolean : **ST_TOUCHES**(Geometry,Geometry)
 - Boolean : **ST_INTERSECTS**(Geometry,Geometry)
 - Boolean : **ST_CONTAINS**(Geometry,Geometry)
 - Geometry : **ST_INTERSECTION**(Geometry,Geometry)
 - Geometry : **ST_GEOMUNION**(Geometry,Geometry)
 - Geometry : **ST_UNION**(Geometry)
 - Geometry : **ST_CONVEXHULL**(Geometry)
 - Geometry : **ST_DIFFERENCE**(Geometry,Geometry)
- postgis.refractory.net oder postgis.net

Wichtige Befehle für die Datenkonvertierung

- iconv 
 - Wandelt Zeichenketten in verschiedene Kodierungen
 - iconv --output=Datei_UTF8.txt --from-code=LATIN9 --to-code=UTF-8 Datei_LATIN9.txt
 - iconv -o Datei_UTF8.txt -f LATIN9 -t UTF-8 Datei_LATIN9.txt
 - LATIN9 entspricht ISO 8859-15 (auch als Latin-9, Westeuropäisch bekannt)

Zeichensatzkonvertierung, Shell



- Setzen der Umgebungsvariable PGCLIENTENCODING bei Programmen wie pgsql2shp, d.h. einem Aufruf von der Kommandozeile (Shell)
 - für **Latin9**:
export PGCLIENTENCODING=LATIN9
 - für **UTF-8**:
export PGCLIENTENCODING=UTF-8



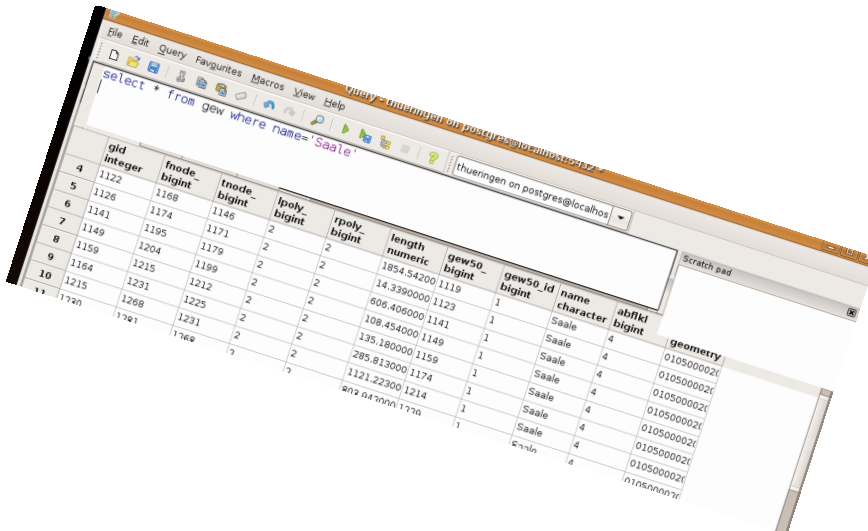
Koordinaten- und Zeichensatzparameter im Kommando shp2pgsql

- Wichtig:
 - Die **Zeichenkodierung** muss bekannt sein
 - Das verwendete **Koordinatensystem** muss bekannt sein
- Beispiel
 - shp2pgsql -W LATIN9 -s 3044 wald.shp wald

Koordinatentransformation in Shapedateien

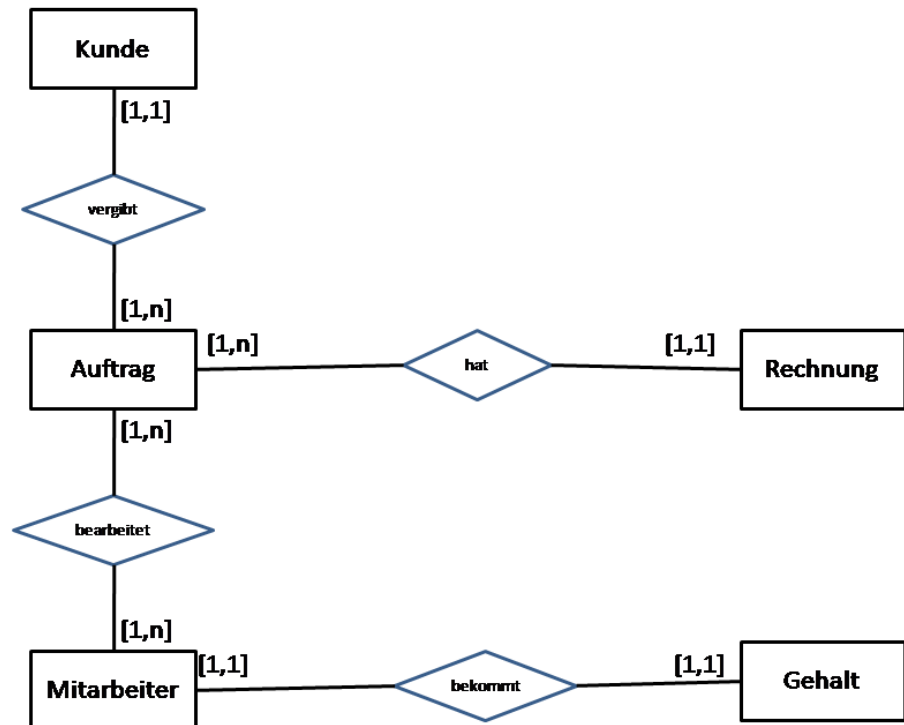
- ogr2ogr, konvertiert Daten in verschiedene Formate und führt Koordinatentransformationen durch
- Beispiel 
 - ogr2ogr -t_srs epsg:25832 -s_srs epsg:4326 wald_epsg25832.shp wald_epsg4326.shp
- Wichtige EPSG Nummern 
 - 25832, ETRS 89, Zone 32
 - 4326, WGS 84
 - 31468, DHDN, Gauss Krüger, Zone 4

- Geodatenbanken – Datenmodellierung

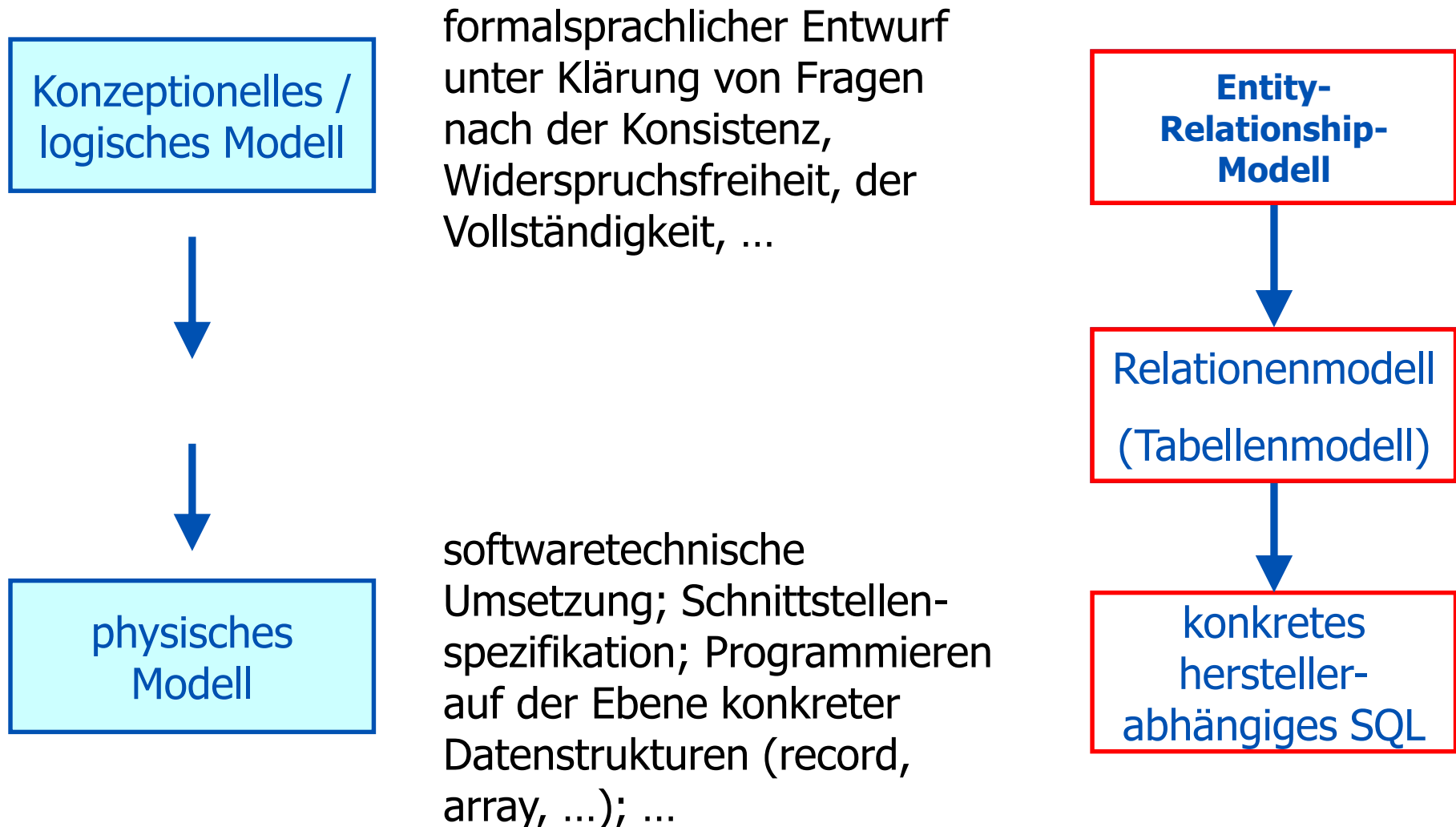


select * from gew where name='Saale'

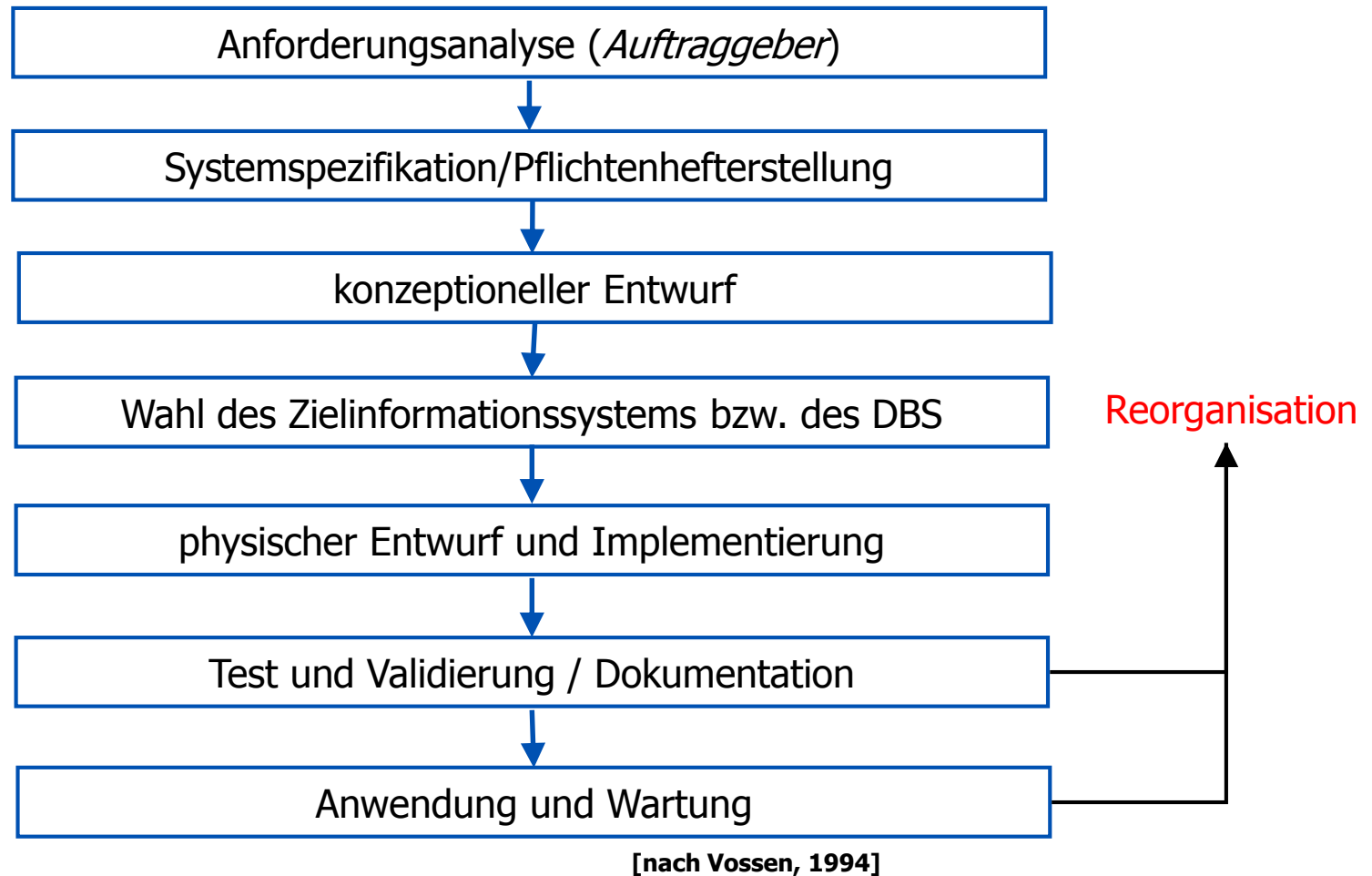
gld	integer	lmode	lnode	lpol	lpol	length	gew50	gew50	name	character	abfkt	geometry
		bigint	bigint	bigint	bigint	numeric	bigint	bigint			bigint	
4	1122	1168	1171	2	2	1854.54200	1119	1	Saale	4	0105000020	
5	1141	1174	1171	2	2	14.3390000	1123	1	Saale	4	0105000020	
6	1141	1174	1171	2	2	606.4060000	1141	1	Saale	4	0105000020	
7	1149	1195	1179	2	2	108.4540000	1149	1	Saale	4	0105000020	
8	1159	1204	1179	2	2	135.1800000	1159	1	Saale	4	0105000020	
9	1164	1215	1212	2	2	285.8130000	1174	1	Saale	4	0105000020	
10	1215	1231	1212	2	2	1121.223000	1214	1	Saale	4	0105000020	
11	1268	1231	1231	2	2	4072.0470000	1170	1	Saale	4	0105000020	



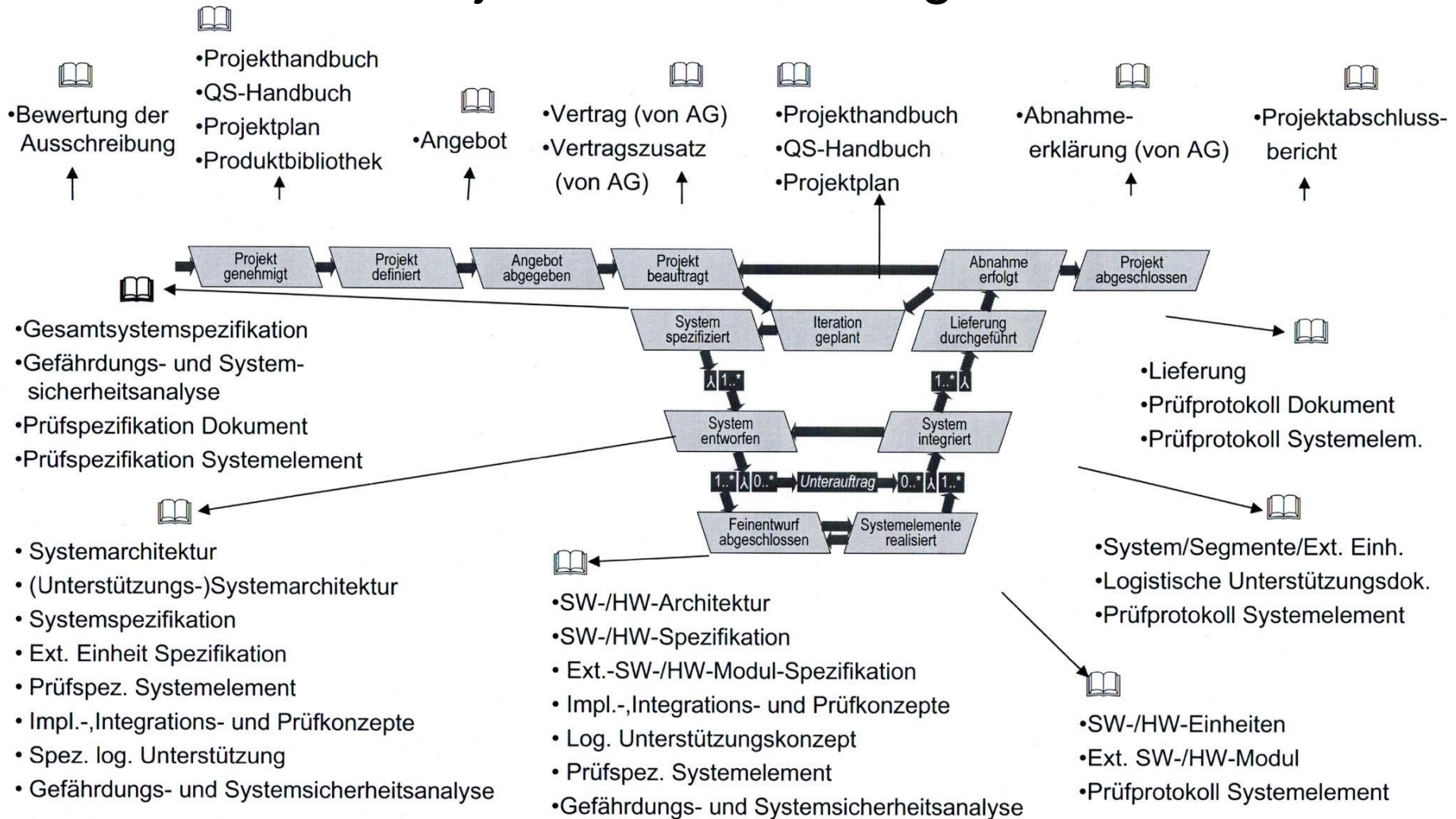
Ebenen der Strukturmodellierung



Phasen des Datenbankentwurfs



V-Modell XT, Entscheidungspunkte / Produkte bei Software Projekten aus Auftragnehmer Sicht



Entity Relationship Modell (ERM)

- Grafische Darstellungsweise der Realwelt
- Unabhängig gegenüber der Wahl des Formats für die logische oder physische Datenmodellierung.
- Entity-Relationship-Modell kann in verschiedene Formate der physischen Datenmodellierung (z.B. in verschiedene SQL-Dialekte) übersetzt werden.

ERM: Entities

- Entities

- unterscheidbare Dinge der realen Welt, z.B. Autos, Personen, Städte
- haben **Attribute** (Eigenschaften/Werte)
 - zugelassene Werte = Wertebereich
 - **Hinweis**: Attribute können auch für die Beschreibung von Relationen genutzt werden.
- besitzen mindestens einen Schlüssel
- können in Beziehung (**Relation**) zueinander stehen

ERM: Attribute

- Beispiel für Attribute von Entitäten:
 - Der Entity-Typ Autor hat die Eigenschaften ‚Vorname‘, ‚Nachname‘, ‚Geburtsdatum‘, ‚Geburtsort‘, ...
- Wertebereiche von Attributen:
 - Aufzählungen (Montag, Dienstag, ...)
 - Bereiche (0 – 9, A – O)
 - Attribut Geburtsort mit Werten der Domäne { Aachen, ... Zerbst}
 - Attribut Vorname vom Typ ‚Zeichenkette‘ mit maximal 100 Zeichen

ERM Beispiel: Bücher einer Bibliothek

- Entities sind Bücher einer Bibliothek
- Entity-Set = Gesamtheit aller Bücher
- Attribute und mögliche Werte:

Attribut	Wertebereich
Buch_NR	„sechsstellige Zahl“
Autor	„Zeichenkette der Länge 60“
Titel	„Zeichenkette der Länge 60“
Rubrik	„Reiseliteratur“, „Belletristik“

Schlüsseleigenschaft und Primärschlüssel

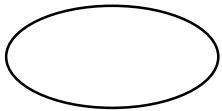
- Schlüssel dienen der **Identifikation** der Datensätze
- zur Identifikation eines Datensatzes gibt es **oft mehrere Schlüssel**, die aus einem oder mehreren Attributen bestehen.
- Festlegung eines **Primärschlüssels** für jede Tabelle → möglichst **minimaler** Schlüssel.
- Schlüssel bilden in den Datenbanken das grundlegende Mittel für die Datensuche: Je länger der Schlüssel, desto größer der Suchaufwand.
- oftmals werden **künstliche** Schlüssel verwendet

ERM: Notation

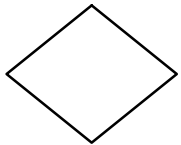
Klassische Schreibweise



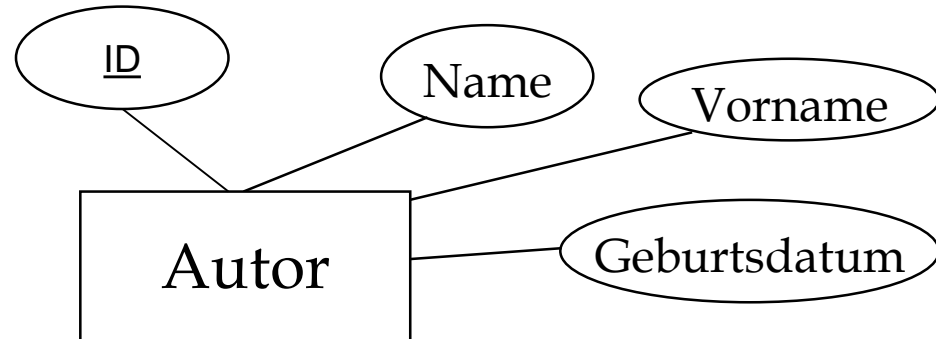
Entity-Typ



Attribut




Relationship/
Beziehung



Attribute werden innerhalb des
Entites dargestellt

Symbol für die Beziehung in
Abhängigkeit der Kardinalität

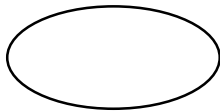
Autor			
<u>ID</u> 	<pi>	1	<M>
Vorname		VA100	
Name		VA100	
Geburtsdatum		D	
Primary Key <pi>			

[Beispiel, Programm DataArchitect]

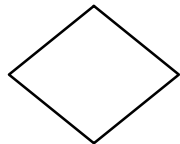
ERM: Notation und Kardinalität



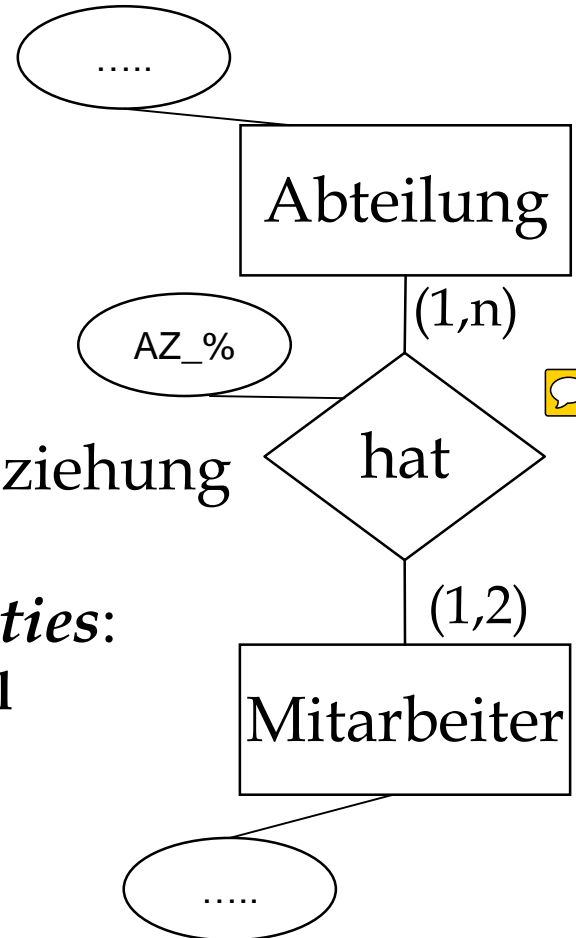
Entity-Typ



Attribut



Relationship/ Beziehung



Beziehungen zwischen Entities:

	zwingend	optional
• eins-zu-eins	$(1,1)$	$(0,1)$
• eins-zu-viele	$(1,n)$	$(0,n)$
• viele-zu-eins	$(n,1)$	$(n,0)$
• viele-zu-viele	(n,m)	

Dabei gilt: n, m sind natürliche Zahlen.

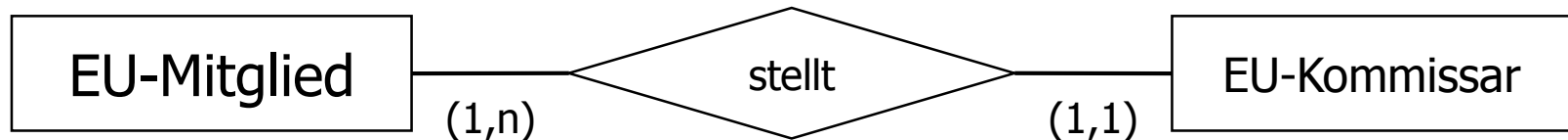
ERM: Beziehungen (1:1)

- **1 : 1 (eins-zu-eins-Beziehung):** Jede Entität der einen Entity-Set steht mit genau einer Entität der anderen Entity-Set in Beziehung.
- Kann oftmals auch direkt in einem Entitytyp dargestellt werden



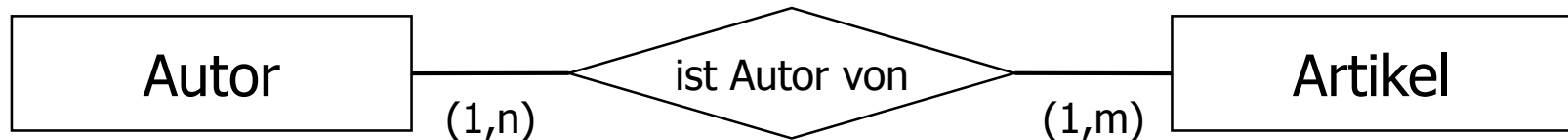
ERM: Beziehungen (1:n)

- **1 : n (eins-zu-n-Beziehung):** Jede Entität der einen Entity-Set steht mit mindestens einer oder mehreren, unbestimmt vielen Entitäten der anderen Entity-Set in Beziehung.



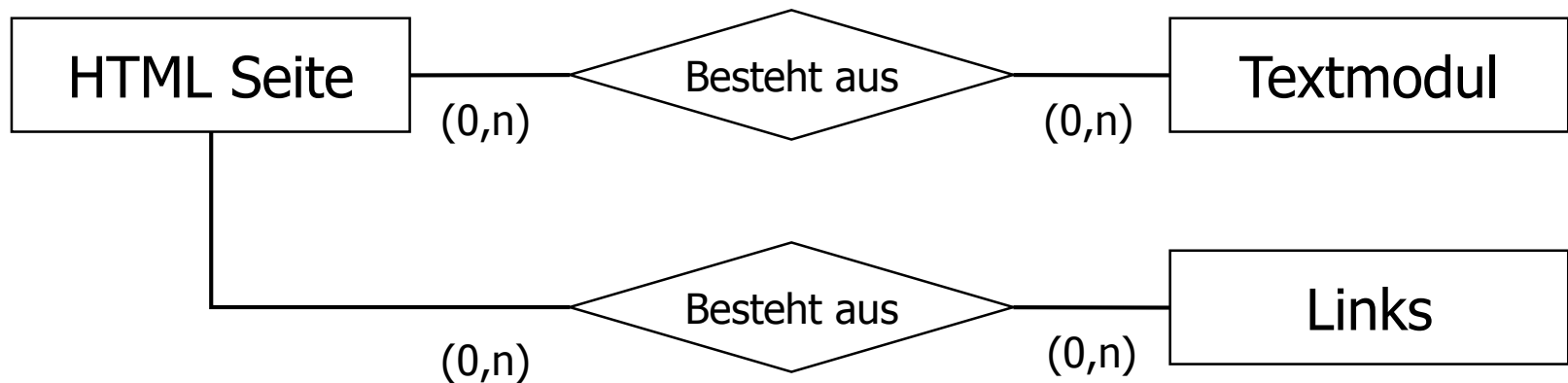
ERM: Beziehungen (n:m)

- **n : m (n-zu-m-Beziehung):** Eine oder mehrere Entitäten der einen Entity-Set stehen mit einer oder mehreren Entitäten der anderen Entity-Set in Beziehung.



ERM: Aggregation

- **Aggregationen (Teil-Ganzes-Beziehungen):**
Eine HTML Seite besteht aus *Textmodulen* und *Links* zu anderen Seiten.



ERM Beispiel: Bibliothek

Eine Bibliothek möchte ihren Buchbestand und die Leser mit Hilfe eines Datenbanksystems verwalten.

Die folgenden Daten sollen für die Bücher der Bibliothek gespeichert werden:

- Autor, Titel, Fachgebiet und Preis.

Die Bücher des Lesesaals können nicht entliehen werden. Die entleihbaren Bücher haben unterschiedliche Entleihfristen (zwei oder vier Wochen). Zu statistischen Zwecken soll außerdem die Häufigkeit der Entleihungen für jedes Buch gespeichert werden. Für jedes entlehene Buch muß der Tag der Ausleihe, das Rückgabedatum und der Entleiher gespeichert werden.

In der Datenbank sollen von den Lesern der Bibliothek die folgenden Daten verwaltet werden:

- Name, Wohnort, Anmeldedatum.

Außerdem soll gespeichert werden, wie oft ein Leser seit seiner Anmeldung Bücher entliehen hat.

Leser können Bücher vorbestellen, der Tag der Vorbestellung und das vorbestellte Buch müssen also ebenfalls verzeichnet werden.

Säumige Leser können mit Strafgebühren belegt werden, falls der zu zahlende Betrag einen bestimmten Wert überschritten hat, so soll der Leser gesperrt werden.

Modellieren Sie das Bibliotheksbeispiel mit Hilfe des ER-Modells

Erzeugen der Datenbanktabellen, Überführung des ER-Modells

- ER-Modell → relationales Datenmodell
 - Normalisierung des ERD
 - Entity-Typen werden direkt auf Tabellen abgebildet
 - für 1:n, n:m Beziehungen zwischen Entity-Typen sind eigene Tabellen notwendig
 - 1:1 Beziehungen werden direkt in den Tabellen dargestellt

Normalformlehre

- Ziel: Redundanzen und Zugriffsprobleme zu verhindern
- durch Normalisierung entstehen neue Relationen (Tabellen)
- Vorgehen: **Entity-fremde** Informationen Erkennen und **Herauslösen**

Unnormalisierte Relationen

KURS						
Kurs_Nr	Kurs_Bez	Halbjahr	Doz_Kürzel	Doz_NN	Doz_VN	Doz_Tel
47	Windows Einführung	2014/2	SR	Raben	Susanne	303851
103	Business-English I	2014/1	WK	Kühnel	Wilfried	684241
142	Windows Einführung	2014/2	SR	Raben	Susanne	303851
144	Excel Einführung	2014/1	AH	Huller	Anna	669221
155	Englisch auf Reisen	2014/1	WK	Kühnel	Wilfried	684241

Lösch-Anomalie:

Löschen von Kurs 144
löscht evtl. einzigen
Eintrag von A.Huller und
damit auch ungewollt die
Dozentin.

Einfüge-Anomalie:

Neue Dozentin Tina
Wolf eintragen ...

Änderungs-Anomalie:

Tel.Nr. von Herrn Kühnel
ändern...

Erste Normalform (1)

P#	NAME	ORT	A#	A-NAME	PR#	PR-NAME
101	Hauck	Karlsruhe	1	Entwicklung	11,12	Intranet, Internet
102	Riester	Rettigheim	2	Personalwesen	13	Dokumentation
103	Rain	Heidelberg	2	Personalwesen	11,12,13	Intranet, Internet, Dokumentation
104	Kammer	Karlsruhe	1	Entwicklung	11, 13	Intranet, Dokumentation



P#	NAME	ORT	A#	A-NAME	PR#	PR-NAME
101	Hauck	Karlsruhe	1	Entwicklung	11	Intranet
101	Hauck	Karlsruhe	1	Entwicklung	12	Internet
102	Riester	Rettigheim	2	Personalwesen	13	Dokumentation
103	Rain	Heidelberg	2	Personalwesen	11	Intranet
103	Rain	Heidelberg	2	Personalwesen	12	Internet
103	Rain	Heidelberg	2	Personalwesen	13	Dokumentation
104	Kammer	Karlsruhe	1	Entwicklung	11	Intranet
104	Kammer	Karlsruhe	1	Entwicklung	13	Dokumentation



Erste Normalform (2)

- Definition der ersten Normalform:
 - Eine Relation ist in **erster Normalform**, wenn alle Attribute nur atomare Werte enthalten und einen Primärschlüssel hat.
 - Zusammengesetzte, mengenwertige oder geschachtelte Wertebereiche sind nicht erlaubt
- Vorteile:
 - Erst durch die erste Normalform werden definierte Abfragen und Sortierungen möglich

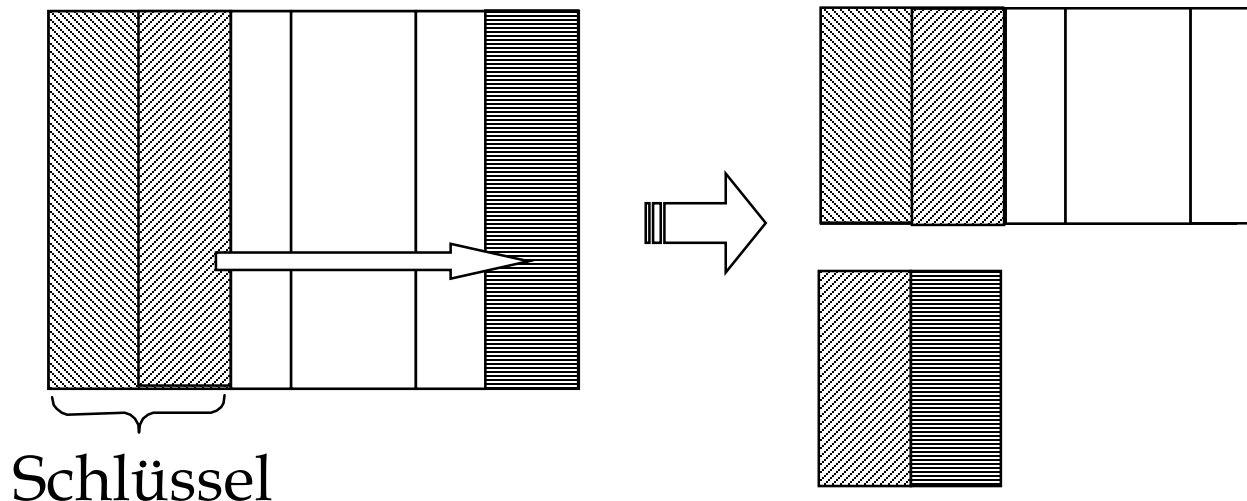
!! Ein Wert pro Zelle !!

Zweite Normalform

- Die **zweite Normalform** betrachtet zusammengesetzte Schlüssel und die Abhängigkeiten der Attribute von diesem zusammengesetzten Schlüssel.
- Eine Relation mit einfachem Schlüssel (d.h. nur aus einem Attribut bestehend), ist damit immer in der zweiten Normalform.

Funktionale Abhängigkeit

- Definition:
 - Ein Attribut Y einer Relation R heißt **funktional abhängig** vom Attribut X derselben Relation, wenn zu jedem X -Wert höchstens ein Y -Wert möglich ist.



Zweite Normalform

- Definition:
 - Eine Relation ist in der **zweiten Normalform**, wenn sie in der ersten ist, und jedes Nichtschlüsselattribut voll funktional abhängig vom Primärschlüssel ist.
- Überprüfung:
 - Sind **alle Nichtschlüsselattribute** immer von **allen Spalten des Primärschlüssels** abhängig?

Beispiel zweite Normalform

Erste
Normalform

P#	NAME	ORT	A#	A-NAME	PR#	PR-NAME
101	Hauck	Karlsruhe	1	Entwicklung	11	Intranet
101	Hauck	Karlsruhe	1	Entwicklung	12	Internet
102	Riester	Rettigheim	2	Personalwesen	13	Dokumentation
103	Rain	Heidelberg	2	Personalwesen	11	Intranet
103	Rain	Heidelberg	2	Personalwesen	12	Internet
103	Rain	Heidelberg	2	Personalwesen	13	Dokumentation
104	Kammer	Karlsruhe	1	Entwicklung	11	Intranet
104	Kammer	Karlsruhe	1	Entwicklung	13	Dokumentation

Zweite Normalform

P#	NAME	ORT	A#	A-NAME
101	Hauck	Karlsruhe	1	Entwicklung
102	Riester	Rettigheim	2	Personalwesen
103	Rain	Heidelberg	2	Personalwesen
104	Kammer	Karlsruhe	1	Entwicklung



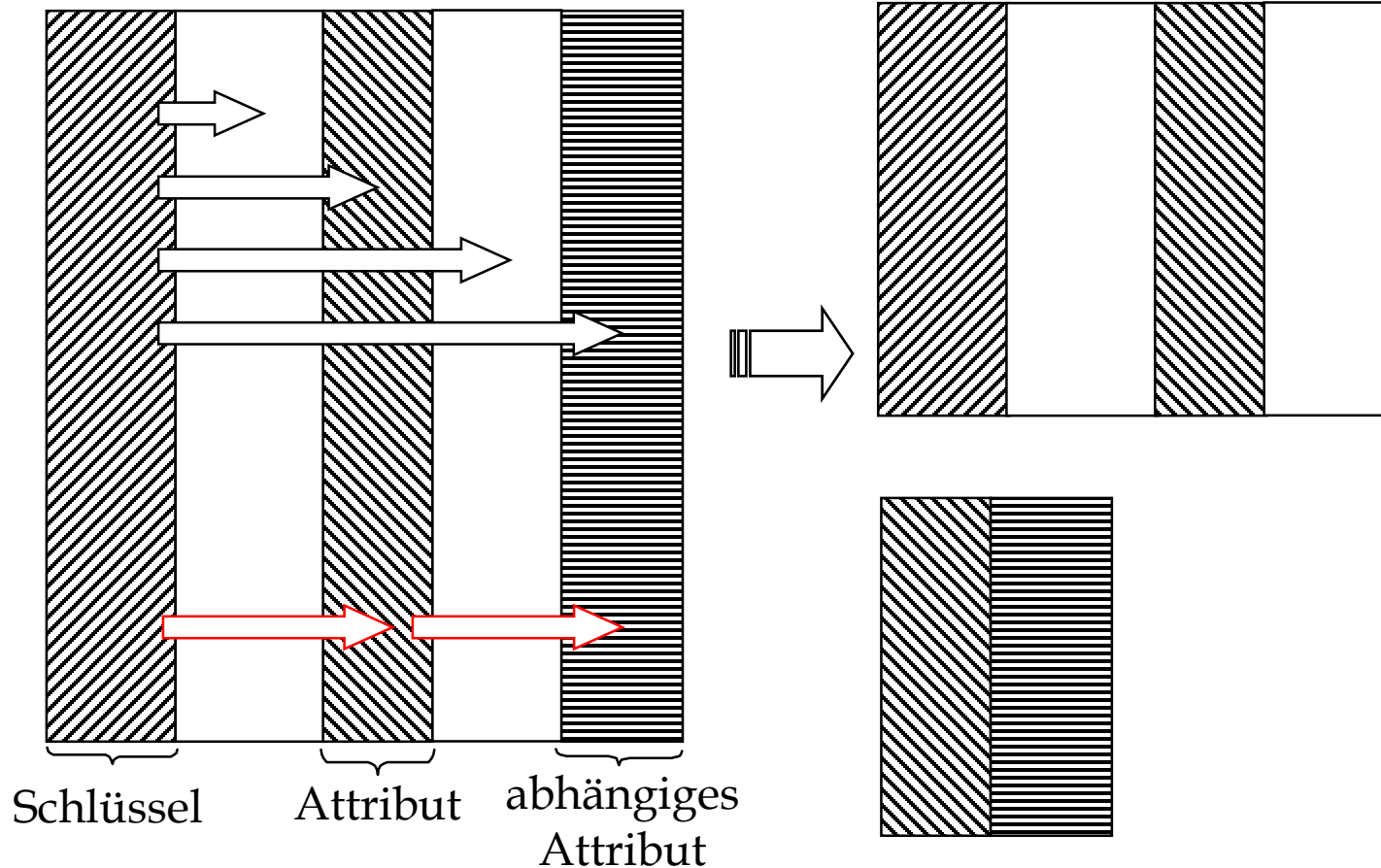
P#	PR#
101	11
101	12
102	13
103	11
103	12
103	13
104	11
104	13

PR#	PR-NAME
11	Intranet
12	Internet
13	Dokumentation

Dritte Normalform

- Sind **alle Nichtschlüsselattribute ausschließlich** vom **Primärschlüssels** abhängig (oder nicht auch von **anderen Nichtschlüsselattributen**)?
- Definition:
 - Eine Relation ist in dritter Normalform genau dann, wenn sie in der zweiten ist und es zwischen Schlüsselattributen und Nichtschlüsselattributen keine transitiven Abhängigkeiten gibt.

Graphische Veranschaulichung zur transitiven Abhängigkeit



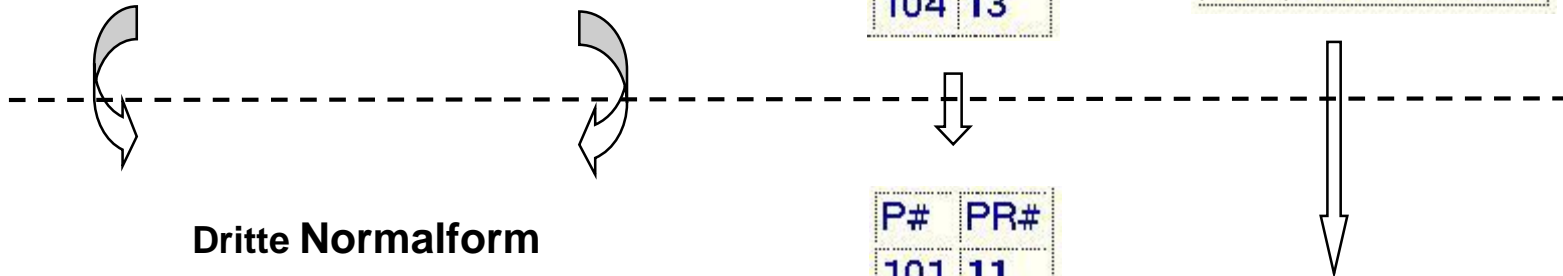
Beispiel für die dritte Normalform

Zweite Normalform

P#	NAME	ORT	A#	A-NAME
101	Hauck	Karlsruhe	1	Entwicklung
102	Riester	Rettigheim	2	Personalwesen
103	Rain	Heidelberg	2	Personalwesen
104	Kammer	Karlsruhe	1	Entwicklung

P#	PR#
101	11
101	12
102	13
103	11
103	12
103	13
104	11
104	13

PR#	PR-NAME
11	Intranet
12	Internet
13	Dokumentation



Dritte Normalform

A#	A-NAME
1	Entwicklung
2	Personalwesen

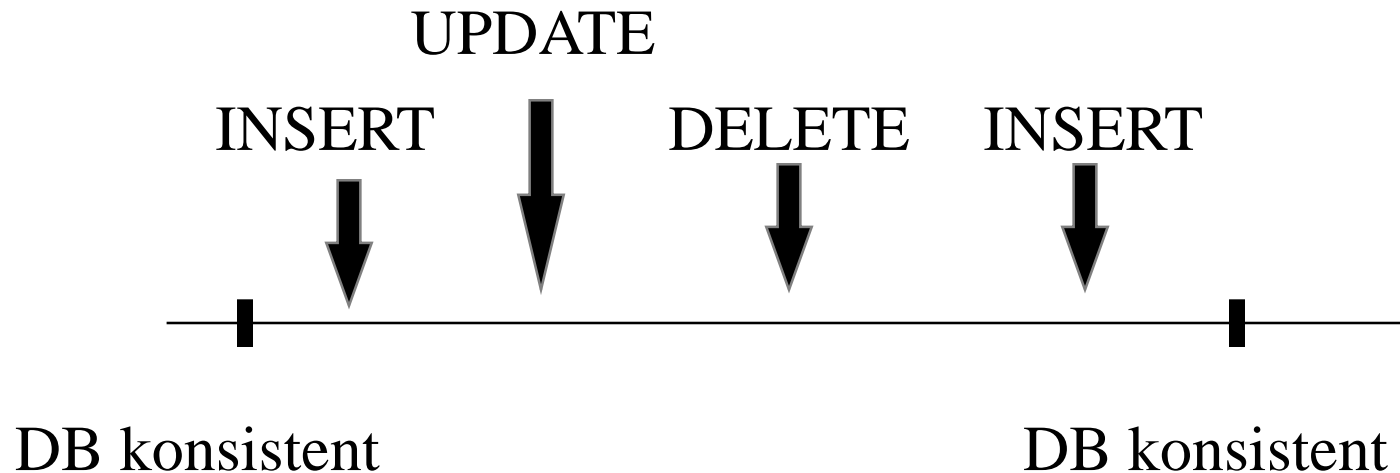
P#	NAME	ORT	A#
101	Hauck	Karlsruhe	1
102	Riester	Rettigheim	2
103	Rain	Heidelberg	2
104	Kammer	Karlsruhe	1

P#	PR#
101	11
101	12
102	13
103	11
103	12
103	13
104	11
104	13

PR#	PR-NAME
11	Intranet
12	Internet
13	Dokumentation

Transaktionen

- Ziel: vor und nach einer Änderung im Datenbestand befindet sich die Datenbank in einem konsistenten Zustand



Transaktionsbeispiele

- Ausleihen eines Buches
 - Ausleihzahl in Buecher und Leser Tabellen erhöhen
 - Satz in der Verleih - Tabelle einfügen
- Buchung für eine Platzreservierung
- Überweisen/Abheben eines Geldbetrages

ACID-Prinzip

- Atomarität (atomicity):
 - "Alles oder nichts"
- Konsistenz (consistency):
 - Einhaltung aller Integritätsregeln
- Isolation:
 - Abgrenzung von anderen Transaktionen
- Dauerhaftigkeit (durability)
 - nach Transaktionsabschluß "überleben"
bei Systemfehlern die geänderten Daten

Befehle zur Transaktionssteuerung (1)

- **BEGIN TRANSACTION**
 - startet eine neue Transaktion
 - Befehlsausführung im Puffer
- **COMMIT TRANSACTION**
 - beendet eine offene Transaktion
 - überträgt geänderte Daten in die Datenbank
 - startet eine neue Transaktion

Befehle zur Transaktionssteuerung (2)

- **ROLLBACK TRANSACTION**
 - macht alle Änderungen seit Beginn der Transaktion rückgängig
 - Datenbankstatus bleibt unverändert!
 - Steuerung durch Nutzer/Programmierer oder das DBMS

Integritätsregeln

- verhindern semantisch unsinnige Datenbankzustände
- wirken auf Spalten, Zeilen, Tabellen oder die gesamte Datenbank
- Reduzierung des Programmieraufwandes
- Verletzung einer Integritätsregel ➡ Zurückweisung der Operation

Arten von Integritätsregeln

- Primärschlüssel, Fremdschlüssel
- NOT NULL Spalte muß immer besetzt sein
- UNIQUE festgelegte Spalten einer Tabelle haben unterschiedliche Werte
- CHECK Überprüfen einer festgelegten Bedingung
- TRIGGER Befehle die unter bestimmten Bedingungen ausgeführt werden

Primär- und Fremdschlüssel (SQL)

- CREATE TABLE abteilung(
 abt_nr CHAR(4) NOT NULL, ...,
 CONSTRAINT abt_pk PRIMARY KEY(abt_nr))
- CREATE TABLE angestellter(
 p_nr INT NOT NULL,...,
 abt_nr CHAR(4),...,
 CONSTRAINT ang_pk PRIMARY KEY(p_nr),
 CONSTRAINT a_nr_fk FOREIGN KEY(abt_nr)
 REFERENCES abteilung)

UNIQUE (SQL)

- Werte in der Tabellenspalte dürfen nicht doppelt vorkommen
- CREATE TABLE bestellung(
 bestell_nr INT UNIQUE,
 art_nr INT REFERENCES artikel)
- CREATE TABLE bestellung(
 bestell_nr INT,
 art_nr INT,
 UNIQUE(bestell_nr),
 FOREIGN KEY(art_nr) REFERENCES artikel)

CHECK (SQL)

- Überprüfen von Bereichsangaben und Bedingungen
- CREATE TABLE angestellter(
 p_nr INT PRIMARY KEY,
 gehalt FLOAT CHECK(
 (*SELECT AVG(gehalt) FROM
angestellter*)<5000 AND
 gehalt BETWEEN 1500 AND 8000),
 geschlecht CHAR(1) CHECK(geschlecht IN('M','W')),
 abt_nr CHAR(4) REFERENCES abteilung)

Eingeschlossene Abfrage im Check: Im Standard SQL definiert, von Oracle und PostgreSQL nicht unterstützt.

CHECK (SQL) Oracle und PostgreSQL

- Überprüfen von Bereichsangaben und Bedingungen, Kombination von Check (Bedingung) und DB Trigger
- CREATE TABLE angestellter(
 p_nr INT PRIMARY KEY,
 gehalt FLOAT CHECK(
 gehalt BETWEEN 1500 AND 8000),
 geschlecht CHAR(1) CHECK(geschlecht IN('M','W')),
 abt_nr CHAR(4) REFERENCES abteilung)

2ter Schritt: definieren des DB-Triggers.

Hinzufügen von Einschränkungen

(Constraints)

- Überprüfen von Bereichsangaben und Bedingungen
- CREATE TABLE angestellter(
 p_nr INT, gehalt FLOAT, geschlecht CHAR(1))
 - ALTER TABLE angestellter ADD CONSTRAINT PK_ANG PRIMARY KEY(p_nr)
 - ALTER TABLE angestellter ADD CONSTRAINT CHECK_ANG_GEHALT CHECK (*gehalt BETWEEN 1500 AND 8000*)
 - ALTER TABLE angestellter ADD CONSTRAINT FK_ANG_ABTEILUNG FOREIGN KEY (abt_nr) REFERENCES abteilung(abt_nr)

Datenbanktrigger

- Sicherung der Datenkonsistenz
- Kann an alle Änderungsfunktionen gekoppelt werden
 - Einfügen, Aktualisieren, Löschen von Datensätzen
- Festlegung, ob pro Datensatz oder pro Änderungsanweisung „gefeuert“ wird
- Trigger können andere Trigger aufrufen

Erzeugen von Triggern

1. Erzeugen einer Trigger-Funktion mit der CREATE FUNCTION Anweisung

- CREATE FUNCTION trigger_function() RETURN trigger AS

2. Binden der Funktion an einen Trigger mit der CREATE TRIGGER Anweisung

- CREATE TRIGGER trigger_name {BEFORE | AFTER } {INSERT
|UPDATE | DELETE}
ON table_name
[FOR [EACH] {ROW | STATEMENT}]
EXECUTE PROCEDURE trigger_function

Beispiel

Erzeugen eines Triggers, der die Veränderung des jeweiligen Gehaltes der Mitarbeiter speichert.

Notwendige *Tabellen* aus Datenbanksicht

- 1) Stammdatentabelle des Mitarbeiters, in der u.a. das aktuelle Gehalt verzeichnet ist. (Tabellenname **Mitarbeiter**)
- 2) Tabelle mit historischen Daten (Daten zur Gehaltsentwicklung → LOG-Datei mit Name des Mitarbeiters, Gehalt, Änderungsdatum), (Tabellenname **Gehalt_Historisch**)



Notwendige *Datenbankobjekte*

- 1) **Datenbanktabellen** mit automatisch generierten ID's
- 2) **Sequenzen**, welche für die ID-Generierung notwendig sind
- 3) **Funktion**, welche bei einer Änderung des Gehaltes aufgerufen wird
- 4) Bindung eines **Triggers** an die Mitarbeiter-Stammdatentabelle

SEQUENZEN

MITARBEITER_ID_SEQ und GEHALT_HISTORISCH_ID_SEQ

1) Sequenz für die Stammdatentabelle der Mitarbeiter

- CREATE SEQUENCE MITARBEITER_ID_SEQ INCREMENT BY 1
MINVALUE 0

2) LOG-Sequenz für die Gehaltsentwicklung (*Gehalt_Historisch*)

- CREATE SEQUENCE GEHALT_HISTORISCH_ID_SEQ INCREMENT
BY 1 MINVALUE 0

Tabellen

Mitarbeiter und Gehalt_Historisch

Mitarbeitertabelle:

```
CREATE TABLE Mitarbeiter(  
  id int4 DEFAULT nextval('MITARBEITER_ID_SEQ ') NOT NULL,  
  vorname varchar(40) NOT NULL,  
  nachname varchar(40) NOT NULL,  
  gehalt int4 NOT NULL  
);
```

LOG-Tabelle

```
CREATE TABLE Gehalt_Historisch (  
  id int4 DEFAULT nextval('GEHALT_HISTORISCH_ID_SEQ') NOT NULL,  
  mitarbeiter_id int4 NOT NULL,  
  letztes_Gehalt varchar(40) NOT NULL,  
  geaendert_am timestamp(6) NOT NULL  
);
```

Funktionsdefinition

log_letzte_gehalts_aenderung()

```
CREATE OR REPLACE FUNCTION log_letzte_gehalts_aenderung()  
  RETURNS trigger AS  
$BODY$  
BEGIN  
  IF NEW.gehalt <> OLD.gehalt THEN  
    INSERT INTO  
      gehalt_historisch(mitarbeiter_id,letztes_gehalt,geaendert_am)  
      VALUES(OLD.id,OLD.gehalt,now());  
  END IF;  
  RETURN NEW;  
END;  
$BODY$  
LANGUAGE plpgsql;
```

Definition des Triggers

trigger_letzte_Gehalts_Aenderung

Erzeugen des Triggers:

```
CREATE TRIGGER trigger_letzte_Gehalts_Aenderung  
  BEFORE UPDATE  
  ON mitarbeiter  
  FOR EACH ROW  
  EXECUTE PROCEDURE func_letzte_gehalts_aenderung();
```

Löschen des Triggers

```
DROP TRIGGER trigger_letzte_Gehalts_Aenderung ON Mitarbeiter;
```

Test der Funktionen

Stammdateneingabe:

```
INSERT INTO mitarbeiter (nachname, vorname, gehalt) values('Köhler','Steffen',3400);  
INSERT INTO mitarbeiter (nachname, vorname, gehalt) values('Löffler','Ralf',2900);
```

Ausgabe der Daten:

```
SELECT * FROM mitarbeiter;
```

ID	NAME	VORNAME	GEHALT
0	Köhler	Steffen	3400
1	Löffler	Ralf	2900

Datenänderung(1)

```
UPDATE mitarbeiter SET gehalt=3500 where nachname='Köhler' AND  
vorname='Steffen'
```

oder

```
UPDATE mitarbeiter SET gehalt=3500 where id=1
```

Datenänderung(2)

```
UPDATE mitarbeiter SET gehalt=3100 where id=2
```