

# Datenmodellierung mit PostgreSQL/PostGIS für QGIS

Dipl. Geogr. Bernhard Ströbl

Anwendungsbetreuer GIS

Kommunale Immobilien Jena

Am Anger 26, 07743 Jena

## Zu mir

- GIS seit über 15 Jahren, QGIS seit fast 10 Jahren
- PostgreSQL/PostGIS seit über 5 Jahren
- Offizielle QGIS-Plugins:
  - DataDrivenInputMask
  - DigitizingTools
- Mehr als 30 Datenmodelle für die Stadtverwaltung Jena (Baulandkataster, Fotoarchiv...)



# Ziele des Workshops

- Tipps aus der Praxis
- Datenmodelle bauen
- Benutzung des QGIS-Plugins *DataDrivenInputMask*
- Theorie nur soweit nötig

# Ausgangslage

- Häufige Kombination QGIS mit Datenhaltung in einer PostgreSQL/PostGIS-Datenbank
- In einer Datenbank werden die Daten **normalisiert** gehalten
- Eine Datenbank stellt die **Integrität** der Daten sicher



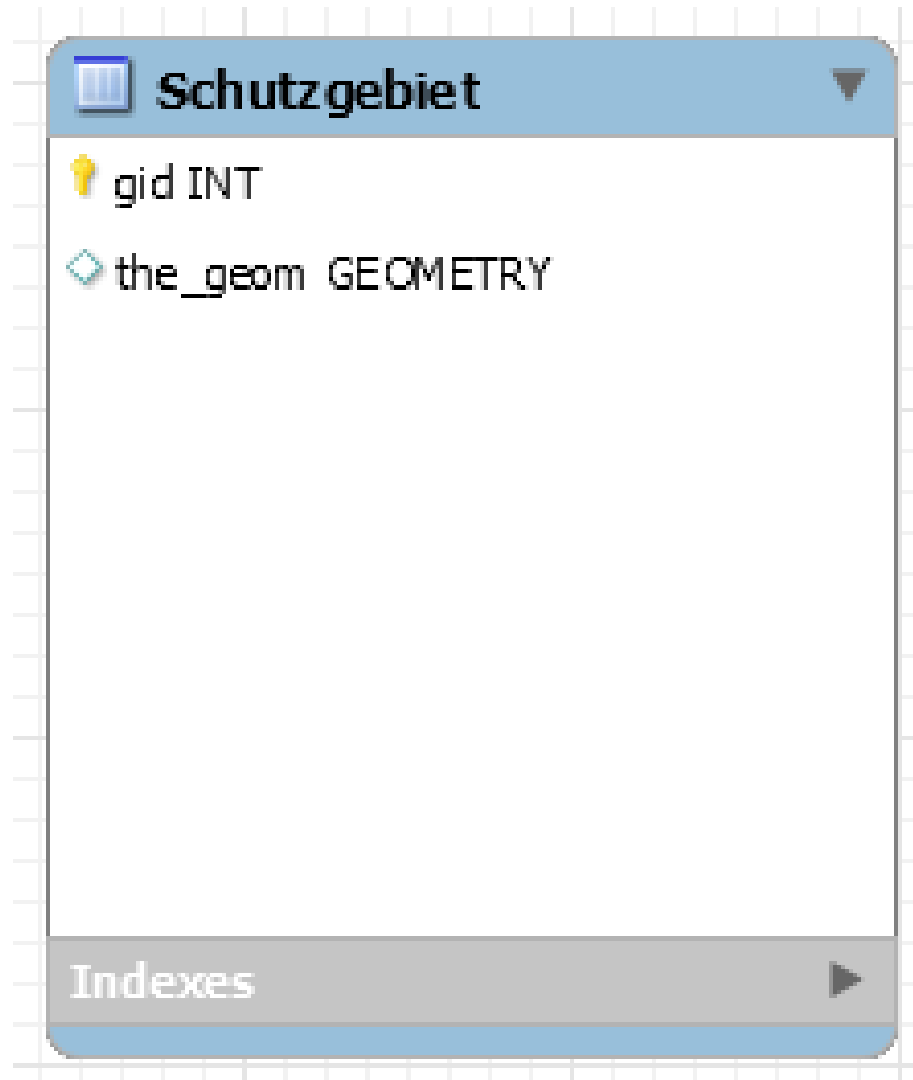
# Normalisierung

- Ziel: Redundanzfreiheit
- Methode: Aufteilen in mehrere Relationen („Tabellen“)

# Normalisierung Voraussetzung

- Erzeugen eines **Primärschlüssels**
  - Üblicherweise **ein** Feld mit entsprechendem Constraint
  - Datentyp INTEGER, oder
  - mit automatisch fortlaufender Numerierung (sequence), Datentyp SERIAL
  - **Niemals** (wirklich niemals) einen „sprechenden“ Schlüssel benutzen, also z.B. Standort + fortlaufende Nummer, denn
  - Sachinformationen gehören **nicht** in den Schlüssel!

# Beispiel Tabelle Schutzgebiet



Quelltext Bereich A

## Beispiel Tabelle Schutzgebiet

- Tabelle hat **ein Schlüsselfeld** mit automatischer Numerierung
- Damit muß sich der Nutzer nicht um die Vergabe von Schlüsselnummern kümmern
- Es ist völlig belanglos, dass, nachdem Datensätze gelöscht wurden, einzelne Schlüsselnummern unbelegt sind, denn
- es handelt sich nicht um eine fortlaufende Numerierung, sondern um einen **Identifikator** des jeweiligen Datensatzes





# Ausgangslage

- Häufige Kombination QGIS mit Datenhaltung in einer PostgreSQL/PostGIS-Datenbank
- In einer Datenbank werden die Daten **normalisiert** gehalten
- Eine Datenbank stellt die **Integrität** der Daten sicher



# Integrität = Daten erfüllen bestimmte (implizite) Annahmen

- Bereichsintegrität
  - Datentyp **Sehr wichtig**
  - Wertebereich **Wichtig**
- Intra-relationale Integrität
  - Eindeutigkeit des Schlüssels **automatisch**
  - Beziehungen zwischen Feldern eines Datensatzes **kann vorkommen**
- Referenzielle Integrität **Sehr wichtig**
  - Beziehungen zwischen Relationen








# Bereichsintegrität

- Datentyp
  - Zahl in Zahlenfeld (auch Jahreszahlen) z.B. INTEGER
  - Text in Stringfeld z.B. VARCHAR
  - Datum in Datumsfeld DATE
  - Ja/Nein in Boolesches Feld BOOL
- Wertebereich
  - CHECK-Constraint
  - Eigener Datentyp; Nachteil: keine Anzeige in QGIS

# Beispiel Tabelle Schutzgebiet








- Soll enthalten:
  - Den Namen des Schutzgebiets
  - Optional eine Schutzgebietsnummer
  - Das Unterschutzstellungsdatum
  - Die in der Schutzgebietsverordnung festgelegte Größe des Schutzgebiets in m<sup>2</sup>
  - Den Schutzgebietstyp (NSG, FFH usw.)

# Beispiel Tabelle Schutzgebiet

Schutzgebiet	
	gid INT
	the_geom GEOMETRY
	gebietsname VARCHAR(64)
	num mer VARCHAR(64)
	unterschutzstellungsdatum DATE
	groesse INT
	gebietstyp VARCHAR(64)
Indexes	

Quelltext Bereiche B und C

# Beispiel Tabelle Schutzgebiet

Schutzgebiet	
	gid INT
	the_geom GEOMETRY
	gebietsname VARCHAR(64)
	num mer VARCHAR(64)
	unterschutzzstellungsdatum DATE
	groesse INT
	gebietstyp VARCHAR(64)
Indexes	

nach 1930  
check constraint








größer 0  
check constraint



# DataDrivenInputMask – CheckConstraint

- DataDrivenInputMask wertet CheckConstraints nicht aus
- Lösung: Konfiguration für QGIS über DataDrivenInputMask
  - Min/Max-Werte
  - Datum mit „today“ oder „today - 30“

# Beispiel Tabelle Schutzgebiet

Schutzgebiet	
	gid INT
	the_geom GEOMETRY
	gebietsname VARCHAR(64)
	num mer VARCHAR(64)
	unterschutzstellungsdatum DATE
	groesse INT
	gebietstyp VARCHAR(64)
Indexes	

sinnvoll?





# Normalisierung Ziel

- Ziel: Redundanzfreiheit
- Methode: Aufteilen in mehrere Relationen („Tabellen“)

# Gebietstyp

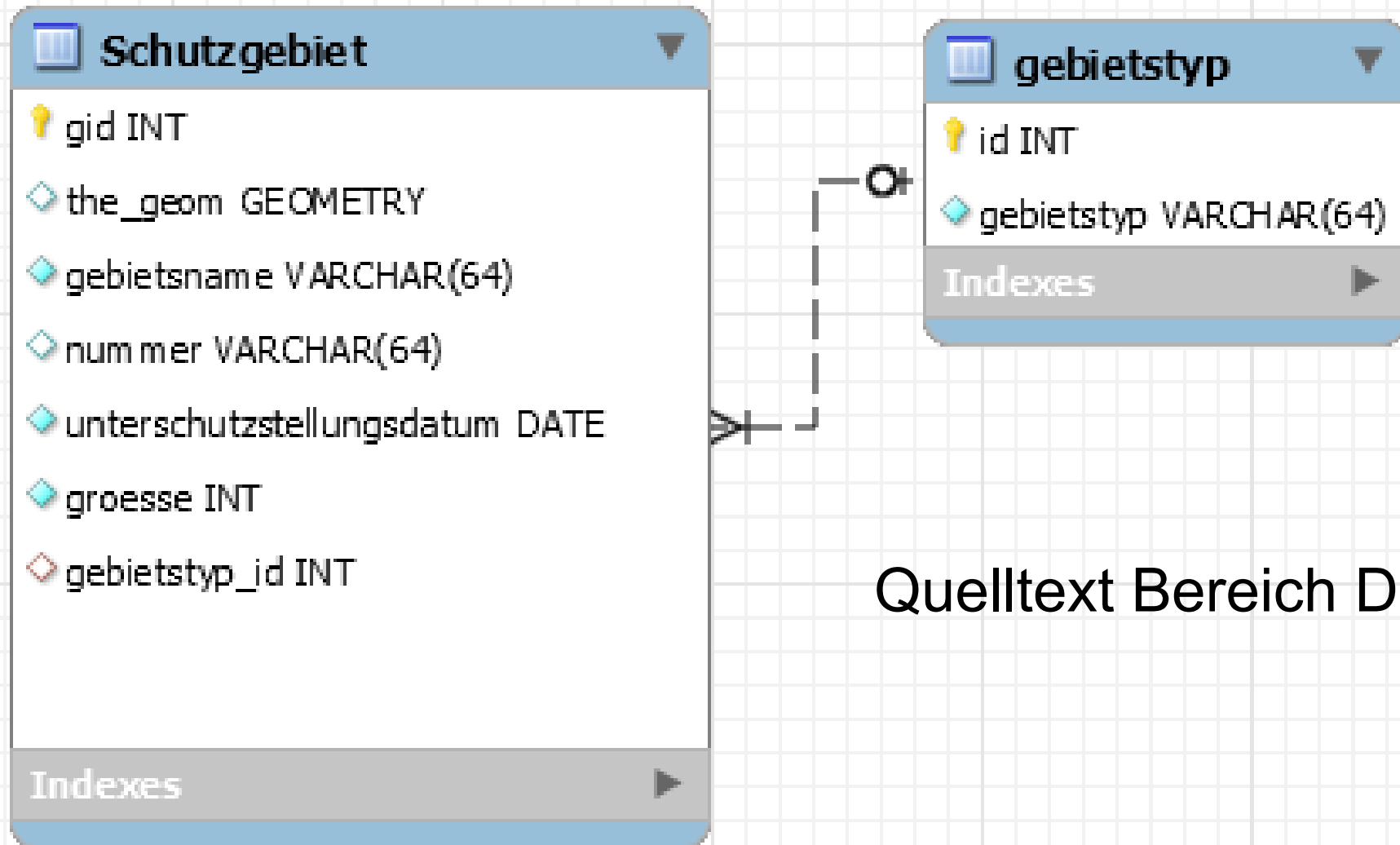
- Eintragung in varchar-Feld erzeugt Redundanz  
=> die Datenbank ist nicht mehr **normalisiert**
- Lösung:
  - Auslagern des Gebietstyps in eine Lookup-Tabelle
  - Referenzieren der Lookup-Tabelle aus der Tabelle Schutzgebiet heraus



# Gebietstyp – referenzielle Integrität

- Es können nur Gebietstypen eingetragen werden, die in der Lookup-Tabelle vorhanden sind
- Definition eines **Fremdschlüssels**  
*Foreign key constraint*

# Gebietstyp – referenzielle Integrität



Quelltext Bereich D

# Gebietstyp – mehrere Gebietstypen pro Schutzgebiet

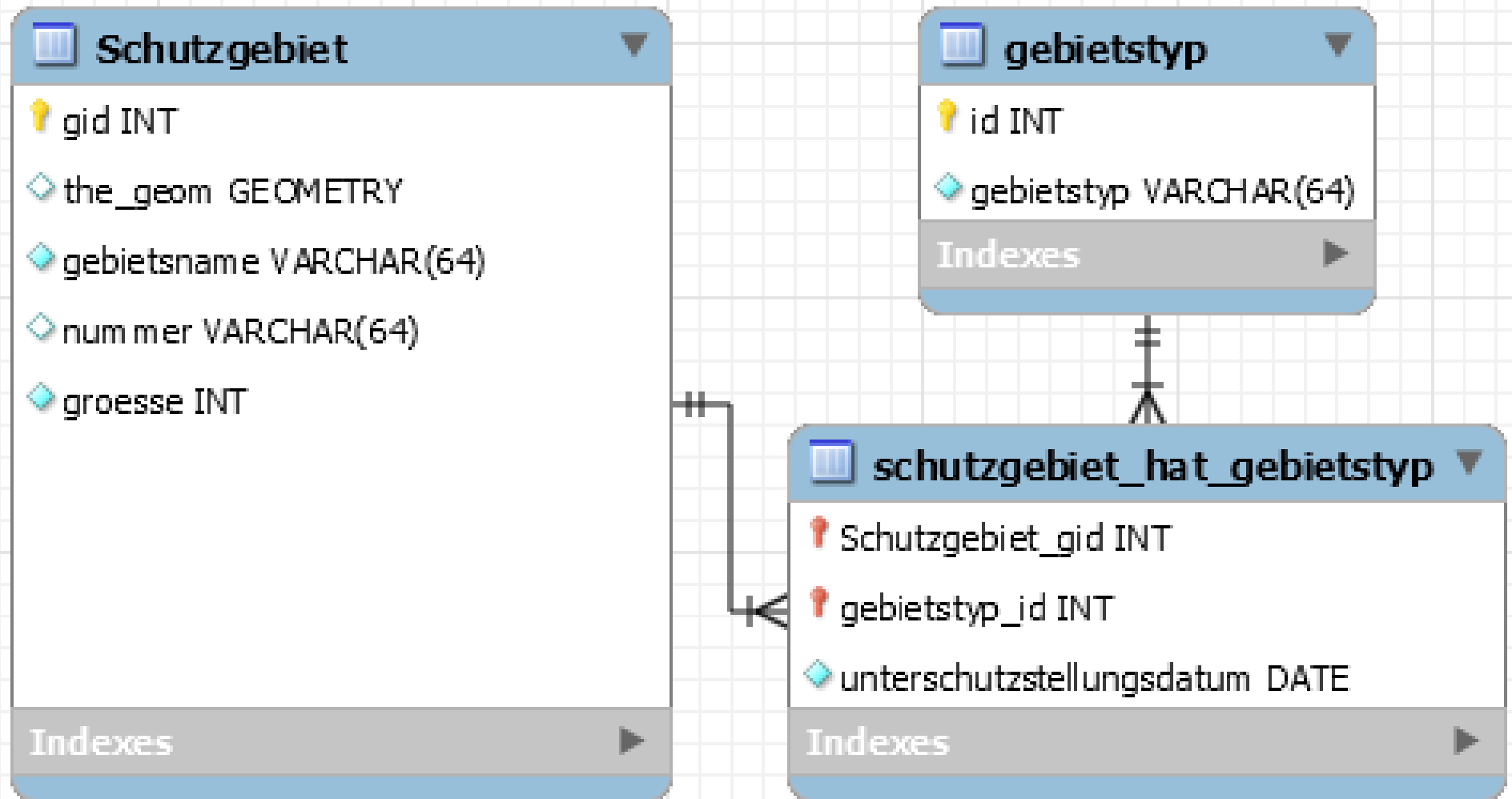


- Sogenannte n-zu-m-Beziehung
  - Ein Schutzgebiet kann mehrere Gebietstypen haben
  - Mehrere Schutzgebiete können den selben Gebietstyp haben
- Wird über eine Zwischentabelle gelöst
- Das Unterschutzstellungsdatum ist abhängig vom Gebietstyp  
=> wandert in die Zwischentabelle

# Gebietstyp – mehrere Gebietstypen pro Schutzgebiet



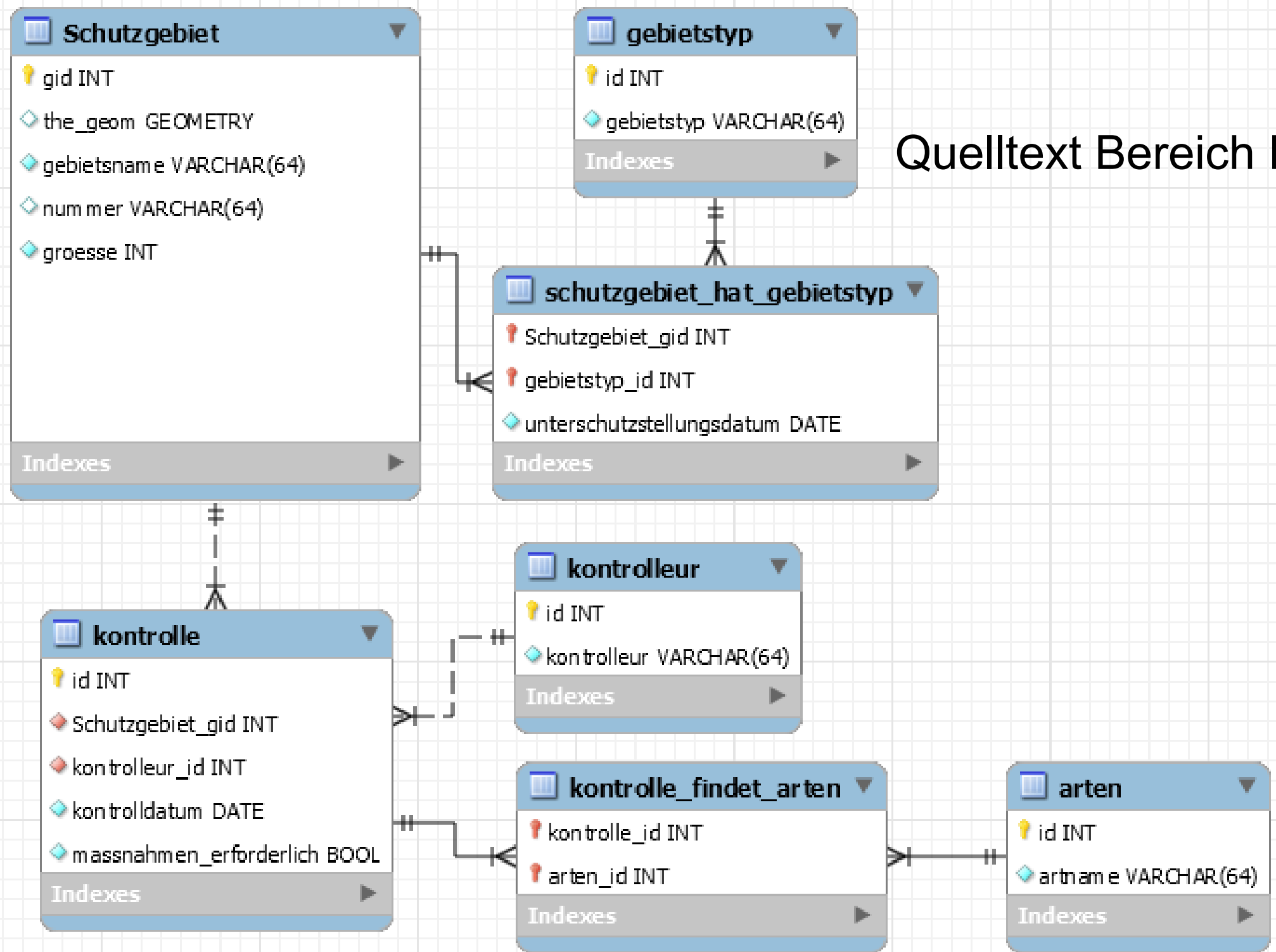
## Quelltext Bereich E





# Gebietskontrollen – bisher Excel-Liste

Schutzgebiet	Kontrolleur	Datum	Artenfunde	Maßnahmen erforderlich
QGISsee	Müller	28.05.2010	QGIS Server, QGIS desktop	ja
Fossgiser Moor	Maier	03.03.2011	geoserver, mapfish	nein
QGISsee	Müller	11.02.2012	QGIS Plugins, QGIS desktop	nein
PostGISwald	Maier	28.03.2012		ja
QGISsee	Schulze	24.04.2013	QGIS WebClient, QGIS Server	nein
PostGISwald	Müller	08.05.2013		nein
Fossgiser Moor	Schulze	05.06.2014	OpenLayers 3, geoserver	ja







# Software

- PostgreSQL <http://www.postgresql.org/>
- PostGIS <http://postgis.refrations.net/>
- QGIS-Plugin DataDrivenInputMask  
<http://plugins.qgis.org/plugins/DataDrivenInputMask/>
- MySQL Workbench (E-R Modelle)  
<http://www.mysql.de/products/workbench/>