Ein leichtgewichtiger, standardorientierter Zugriffsmechanismus auf Geometrieobjekte in Geodatenbanksystemen

THOMAS BRINKHOFF[1]

Zusammenfassung: Trotz der Standardisierung der Datenmodelle ist der Zugriff von Anwendungsprogrammen auf in Datenbanksystemen gespeicherten Geodaten nicht vereinheitlicht, so dass man auf proprietäre Lösungen der Datenbankhersteller angewiesen ist oder komplexe Frameworks nutzen muss. Gerade für kleinere Anwendungen stellt der zweite Ansatz aufgrund des Overheads keine geeignete Lösung dar. Somit besteht Bedarf an einem leichtgewichtigen Zugriffsmechanismus, der sich an standardisierten Datenbankzugriffsschnittstellen orientiert und Geometrietypen in einheitlicher Weise inkludiert. Der Beitrag stellt eine konzeptionelle Erweiterung der Datenbankzugriffsschnittstelle JDBC für Geometriedatentypen vor, die den einfachen Zugriff auf Geometrieobjekte gemäß dem Simple-Feature-Modell, auf Grafikobjekte sowie auf Textrepräsentationen erlaubt. Erprobt wurde der Ansatz anhand einer Anwendung, die die textuelle und grafische Anzeige unterschiedlicher Geometrietypen unterstützt.

1 Einleitung

Geodatenbanksysteme stellen ein wesentliches Werkzeug zur offenen, interoperablen Nutzung von Geodaten dar, da sie es beliebigen Anwendungsprogrammen erlauben, auf Geometrieobjekte unter Erhalt ihrer Semantik zuzugreifen (BRINKHOFF, 2013). Dazu besitzt das Datenbanksystem eine Datenbankzugriffsschnittstelle. Eine solche Schnittstelle umfasst u.a. Operationen, um Tabellenstrukturen und Daten abfragen, einfügen, verändern und löschen zu können. Um deren Nutzung zu vereinfachen, werden standardisierte Datenbankzugriffsschnittstellen angeboten. Bekannte Standards sind ODBC (Open Database Connectivity) für Windows und JDBC (Java Database Connectivity) für die Java-Plattform[2].

Obwohl die Standardisierung der Datenmodelle für Geodatenbanken durch ISO 19125 (ISO, 2004) und ISO SQL/MM Spatial (ISO/IEC, 2011) weit fortgeschritten ist, hat dies bislang keinen Niederschlag in den standardisierten Datenbankzugriffsschnittstellen gefunden. Stattdessen bieten die Datenbankhersteller systemspezifische Bibliotheken (APIs) an, die es – mehr oder weniger komfortabel – erlauben, Geometrieobjekte aus Datenbankanfragen zu erhalten bzw. diese abzuspeichern. Bei PostGIS ist dies für die Programmiersprache Java das Paket `org.postgis`[3] und bei Oracle die Oracle Spatial and Graph Java API[4]. Die direkte Nutzung

[1] Thomas Brinkhoff, Jade Hochschule Oldenburg, Institut für Angewandte Photogrammetrie und Geoinformatik, Ofener Str. 16/19, 26121 Oldenburg, E-Mail: thomas.brinkhoff@jade-hs.de

[2] ORACLE CORP.: JDBC Overview, http://www.oracle.com/technetwork/java/overview-141217.html

[3] Dokumentation: http://postgis.refractions.net/documentation/javadoc/org/postgis/package-summary.html

[4] Dokumentation: http://docs.oracle.com/cd/E16655_01/appdev.121/e20856/index.html

solcher Bibliotheken hat Systemabhängigkeiten im Programmcode zur Folge; Änderungen in den APIs schlagen entsprechend durch.

Eine grundsätzliche Lösung für diese Problematik ist die Nutzung von Frameworks, die den Zugriff auf die Datenbank kapseln. Einen allgemeinen Standard für objektorientierte Datenbank-Frameworks stellt das Java Persistence API (JPA) dar, das u.a. von EclipseLink und Hibernate realisiert wird (MÜLLER & WEHR, 2012). Für die Bereitstellung von Geoobjekten gemäß OGC GeoAPI[5] bietet das Java-basierte Open Source GIS-Toolkit GeoTools die `DataStore`-Schnittstelle an, mit deren Hilfe in einheitlicher Art und Weise u.a. auf Dateien (z.B. ESRI Shapefiles), WFS-Dienste und Geodatenbanken zugegriffen werden kann[6].

Gerade für kleinere Anwendungsprogramme, die auf Geodaten aus Geodatenbanksystemen zugreifen müssen, stellt die Nutzung komplexer Frameworks keine befriedigende Lösung dar, da der Overhead bezüglich Einarbeitung und notwendigem Programmcode oftmals in keinem Verhältnis zur benötigten Funktionalität steht. So besteht GeoTools 10.4 aus 180 `jar`-Dateien im Gesamtumfang von 60 MB. Somit besteht Bedarf an einem leichtgewichtigen Zugriffsmechanismus, der standardisierte Datenbankzugriffsschnittstellen um die Unterstützung von Geometrietypen erweitert. Dieser Beitrag stellt nachfolgend eine solche Erweiterung der JDBC-Datenbankzugriffsschnittstelle vor.

Der Beitrag gliedert sich wie folgt: Der nächste Abschnitt erläutert die Erweiterung der JDBC-Datenbankzugriffsschnittstelle. In Abschnitt 3 wird der „Spatial Database Viewer“ als Anwendung vorgestellt, mit der die Erweiterungen für Oracle und PostgreSQL/PostGIS erprobt wurden. Der Beitrag schließt mit einer Zusammenfassung und einem Ausblick auf künftige Arbeiten.

2 Erweiterung von JDBC für einen Geometriezugriff

2.1 JDBC

JDBC ist Teil der Java-Klassenbibliothek (JDK). Es stellt eine Reihe von Schnittstellenklassen im Paket `java.sql` zur Verfügung, die den Zugriff auf Datenbanken ermöglichen[7]. Diese Schnittstellen werden durch Klassen implementiert, die durch einen sogenannten JDBC-Treiber für ein konkretes Datenbanksystem bereitgestellt werden. Der Treiber meldet sich bei dem Treibermanager (Klasse `DriverManager`) an; von diesem erhält die Anwendung das Objekt, das die Datenbankverbindung repräsentiert (Schnittstelle `Connection`). Objekte für SQL-Anweisungen stellt das `Connection`-Objekt bereit. Folgende Schnittstellen für Anweisungen existieren:

[5] Da dieser Teil der GeoAPI nicht in der vom OGC verabschiedeten Version 3.0 (CUSTER, 2011) enthalten ist, siehe: http://www.geoapi.org/snapshot/pending/index.html

[6] Dokumentation: http://docs.geotools.org/latest/userguide/library/data/datastore.html

[7] Dokumentation: http://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html

- `Statement` dient zur Formulierung von normalen SQL-Anweisungen.
- `PreparedStatement` erlaubt die Formulierung von vorbereiteten SQL-Anweisungen, bei denen man vor Durchführung einer konkreten Anfrage variable Werte über Parameter belegt.
- `CallableStatement` ruft Prozeduren und Funktionen auf, die in der Datenbank gespeichert sind.

Führt man eine Anfrage über ein Anweisungsobjekt durch, erhält man ein `ResultSet`-Objekt, das den Datenbank-Cursor darstellt. Mit dessen Hilfe kann man über das Anfrageresultat iterieren. Abb. 1 stellt die genannten Klassen und Schnittstellen in einer Übersicht vor.

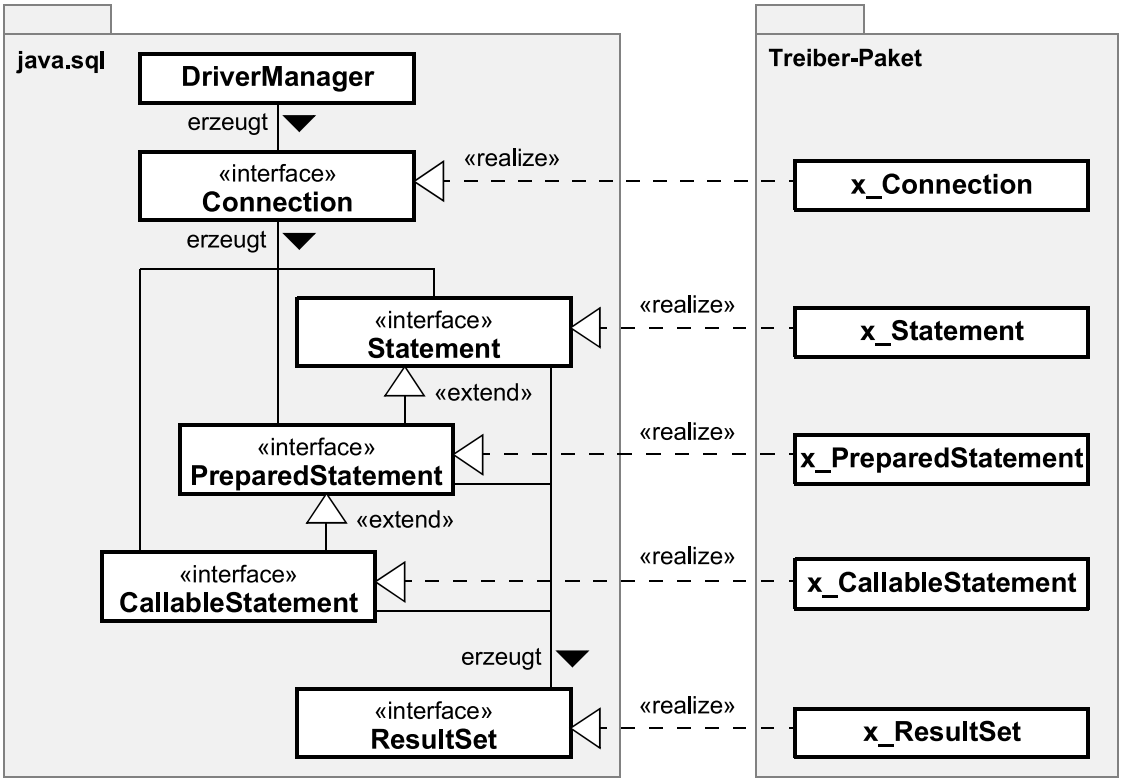


Abb. 1: JDBC-Klassen und -Schnittstellen (Auswahl)

Die Schnittstellen `PreparedStatement`, `CallableStatement` und `ResultSet` erlauben es, Parameter- bzw. Attributwerte über `get`-Methoden abzufragen bzw. über `set`- bzw. `update`-Methoden zu setzen. Dies erfolgt typabhängig (z.B. `get`/`set`/`updateInt` für ganze Zahlen und `get`/`set`/`updateString` für Zeichenketten). Für Geometrietypen stehen keine entsprechenden Methoden zur Verfügung.

2.2 Proprietärer Geometriezugriff

Geodatenbanksysteme stellen Geometrieobjekte über die JDBC-Schnittstelle zur Verfügung. Ein Zugriff ist dem Grundsatz nach über die Methoden `getObject` bzw. `set`/`updateObject` möglich, die jeweils ein Java-`Object` zurückgeben bzw. erwarten. Proprietäre APIs erlauben die Interpretation bzw. komfortablere Nutzung eines solchen Objekts. Beim PostGIS-Paket

`org.postgis` wird dazu für den SQL-Datentyp `GEOMETRY` (POSTGIS, 2013) die Klasse `PGgeometry` bereitgestellt. Bei Oracle korrespondiert zum SQL-Datentyp `SDO_GEOMETRY` die Klasse `java.sql.Struct`, die dessen objektrelationale Struktur gemäß JDBC repräsentiert (ORACLE CORP., 2013). Über die Oracle Spatial and Graph API kann der `Struct` in eine `oracle.spatial.geometry.JGeometry` umgewandelt werden kann.

2.3 Strukturelle Anpassung der Schnittstellen

Für einen an JDBC orientierten Zugriff auf Geometrien müssen die drei `Statement`-Schnittstellen und `ResultSet` erweitert werden. Dazu wurden die Schnittstellen `SpatialStatement`, `SpatialPreparedStatement`, `CallableSpatialStatement` und `SpatialResultSet` entworfen, die die JDBC-Schnittstellen um zusätzliche Methoden ergänzen (vgl. Abs. 2.4). Implementiert werden diese Schnittstellen durch entsprechende `DefaultXXX`-Klassen, die über den Konstruktur das korrespondierende JDBC-Objekt erhalten. Methodenaufrufe nach JDBC werden an dieses Objekt weitergeleitet, während die geometrischen Zugriffe an datenbankspezifische Konverter-Klassen delegiert werden. Da `Statement`-Objekte von dem `Connection`-Objekt bereitgestellt werden, muss dieses um Methoden ergänzt werden, die die erweiterten Objekte zurückgeben. Dazu erweitert die Schnittstelle `SpatialConnection` die JDBC-`Connection`; eine abstrakte Klasse `AbstractSpatialConnection` realisiert datenbankunabhängige Funktionalität. Die Bereitstellung instanziierbarer `SpatialConnection`-Klassen erfolgt in spezifischen Paketen, da datenbankabhängige Konverter-Objekte den `Statement`-Konstruktoren übergeben werden.

Abb. 2 zeigt die Struktur im Überblick. Im Paket `spatialsql` befinden sich die systemunabhängigen Schnittstellen und Klassen und in `spatialsql.x` die Klassen für ein spezifisches Geodatenbanksystem *x*. Aus Übersichtsgründen sind die Schnittstellen und Klassen für `SpatialPreparedStatement` und `CallableSpatialStatement` nicht dargestellt; sie betten sich wie `SpatialStatement` ein. Gleiches gilt für die nicht dargestellten Konverter-Klassen; sie ergänzen das Modell in analoger Form zu `GeometryConversion` (vgl. Abschnitt 2.4).

2.4 Konkrete Anpassung der Schnittstellen

Gemäß dem geschilderten Ansatz wurde der Zugriff auf Geometrieobjekte gemäß dem Simple-Feature-Modell, auf Grafikobjekte sowie auf standardisierte Textrepräsentationen realisiert.

2.4.1 Well-Known Text

Für das Simple-Feature-Modell ist mit “Well-Known Text” eine standardisierte Textrepräsentation definiert (ISO, 2004) (ISO/IEC, 2011). Für den entsprechenden Zugriff stehen in den erweiterten Schnittstellen `getWKT`-Methoden zur Verfügung. Die Umwandlung erfolgt in einer datenbankspezifischen `WKTConversion`-Klasse. Oracle Spatial and Graph Java API unterstützt diese Umwandlung (mit Einschränkungen) durch die Klasse `oracle.spatial.util.WKT`. Die Einschränkungen betreffen zusammengesetzte 3D-Flächen und 3D-Körper, die im Datentyp `SDO_GEOMETRY` vorliegen können. Für den erstgenannten Fall wird die Problematik entschärft, indem die Bestandteile in 2D-Geometrien überführt und diese zu einer `GeometryCollection` zusammengeführt werden.

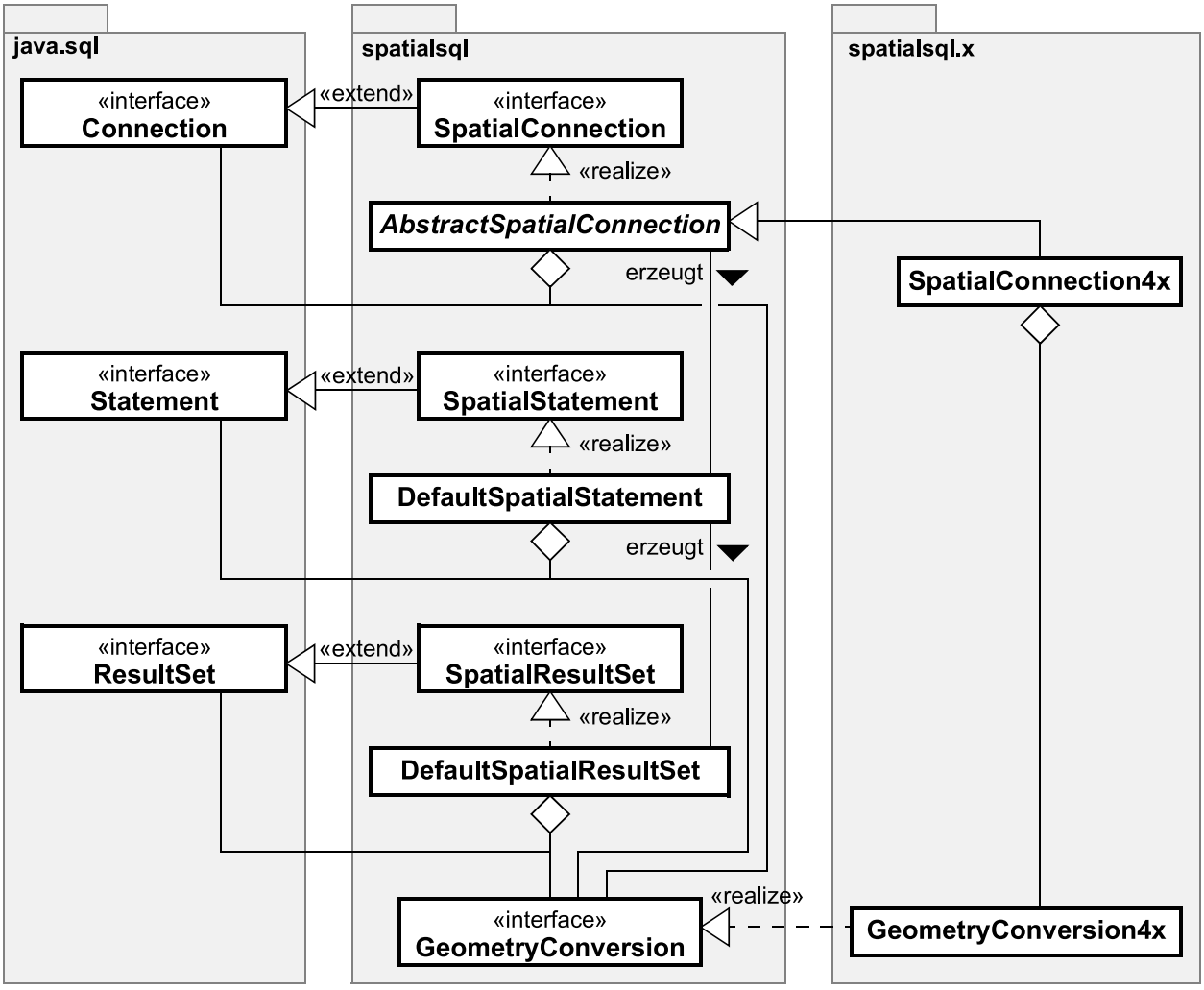


Abb. 2: Erweiterung der JDBC-Schnittstellen (Auswahl)

2.4.2 Simple-Feature-Modell

Für die Überführung in Geometrien des Simple-Feature-Modells benötigt man zunächst ein entsprechendes Datenmodell in der genutzten Programmiersprache. Da die OGC GeoAPI bislang keine entsprechenden Schnittstellen definiert hat, ist man auf andere Anbieter angewiesen: Die JTS Topology Suite stellt ein entsprechendes API bereit (http://sourceforge.net/projects/jts-topo-suite/). Für die Übergabe von Geometrien werden die Schnittstellen um `setGeometry`- bzw. `updateGeometry`-Methoden ergänzt. Für die Anfrage von Geometrien werden neben `getGeometry` über `getPoint`, `getLineString` usw. auch Zugriffsmethoden für `Geometry`-Unterklassen angeboten. Die Umwandlung erfolgt in einer datenbankspezifischen `GeometryConversion`-Klasse.

2.4.3 Grafische Primitive

Grafische Primitive des Abstract Window Toolkits (AWT) sind in Java im Paket `java.awt.geom` zusammengefasst[8]. Für die Abfrage von minimal umgebenden Rechtecken stehen in `SpatialResultSet` `getMBR`-Methoden zur Verfügung, die jeweils ein `Rectangle2D` liefern. Punktförmige grafische Primitive können in Java durch die Klasse `Point2D` repräsentiert werden, Linien u.a. durch `Line2D` und `Path2D` und für Flächen steht die `Area`-Klasse zur Verfügung. Zur Abfrage dieser grafischen Primitive wurde der `SpatialResultSet` um `getShape`-Methoden ergänzt. Alle Primitive außer `Punkt2D` erfüllen die Schnittstelle `java.awt.Shape`; eine Klasse oder Schnittstelle, die alle Primitive erfüllen, existiert neben der allgemeinen `Object`-Klasse leider nicht. Daher ist `Object` der Rückgabetyp der `getShape`-Methoden. Die Umwandlung erfolgt in einer datenbankspezifischen `ShapeConversion`-Klasse. Im Fall von Oracle werden 3D-Geometrien auf 2D-Grafikprimitive reduziert und NURBS durch Streckenzüge approximiert.

2.5 Erweiterungen

Geodatenbanksysteme stellen oftmals Datentypen zur Verfügung, die den Standard-Geometriedatentyp ergänzen. Bei Oracle Spatial stellt die `ST_GEOMETRY`-Klasse einen solchen Datentyp dar, die eine standardkonforme, objektorientierte Repräsentation von Geometrien ermöglicht. Bei PostGIS ist der Datentyp `GEOGRAPHY` zu nennen, der bezüglich geografischer Koordinaten sphärische Abstands- und Flächenberechnungen unterstützt. Beide Datentypen werden nicht direkt von den proprietären APIs abgedeckt, so dass die Konvertierungsklassen entsprechende Objekte spezifisch detektieren und umwandeln müssen. Dies kann ggf. zu Leistungseinbußen führen.

3 Anwendung

Die vorgestellten JDBC-Erweiterungen wurden in der Anwendung „Spatial Database Viewer“ genutzt, die es erlaubt, per SQL Anfragen auf Geodatenbanken zu stellen und die Ergebnisse tabellarisch, grafisch und als WKT-Liste zu visualisieren[9]. Das Programm steht in zwei Versionen zur Verfügung, die den Zugriff auf Oracle Spatial bzw. PostgreSQL/PostGIS erlauben. Neben vektorbasierten Geometrien und deren Metadaten unterstützt der Viewer (falls vom Datenbanksystem angeboten):

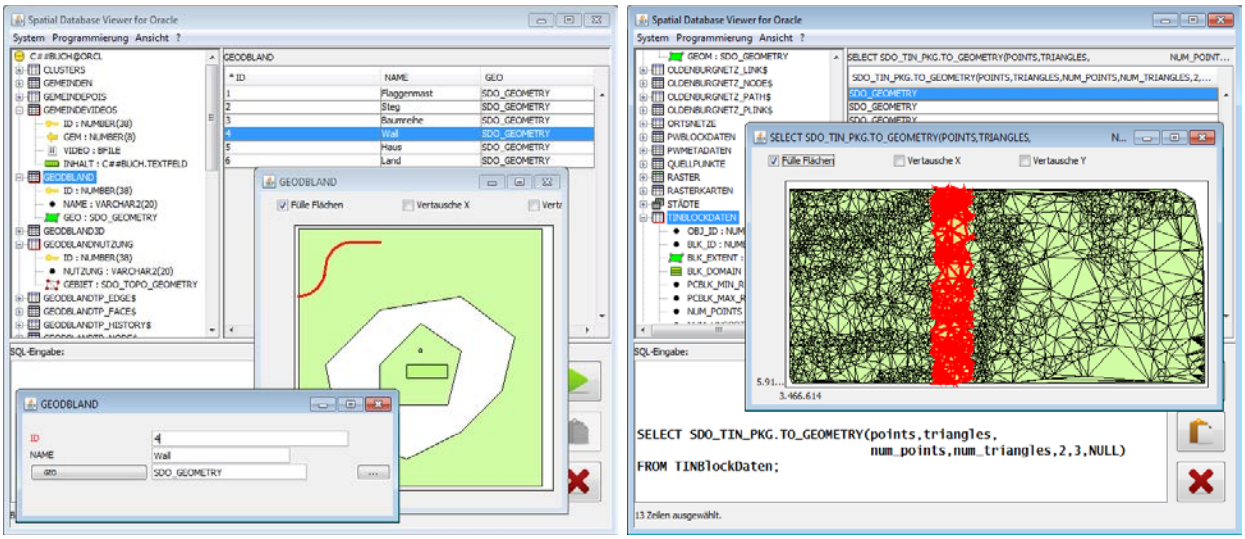
- objektrelationale Merkmale wie Felder, Objekte, prozedurale Erweiterungen zur Programmierung von Methoden, Objekttabellen, Referenztypen und Objektsichten,
- RowID-, CLOB- und XML-Attribute,
- Rasterdatentypen und
- topologische Attribute und deren Metadaten (z.Zt. nur Oracle Spatial).

[8] Dokumentation: http://docs.oracle.com/javase/7/docs/api/java/awt/geom/package-summary.html
[9] Der Spatial Database Viewer kann frei unter http://www.geodbs.de/ heruntergeladen werden.

Durch die Verwendung der erweiterten JDBC-Klassen können große Teile des Programmcodes in beiden Anwendungen identisch verwendet werden. Auch der Umfang des Programms (als Jar-Dateien) kann als kompakt bezeichnet werden:

- Anwendung: ~200 KB (wovon ~65 KB für Icons)
- JDBC-Erweiterung für Geometrieobjekte: ~60 KB
- JDBC-Treiber: ~3,2 MB (Oracle) bzw. ~460 KB (PostgreSQL)
- Proprietäre JDBC-Geobibliotheken: ~420 KB (Oracle) bzw. ~75 KB (PostGIS)
- Raster-Visualisierung (inkl. XML-Unterstützung): ~3,7 MB (Oracle) bzw. 0 MB (PostGIS)

Abb. 3 zeigt anhand von Screenshots die Nutzung der Funktionalität der JDBC-Erweiterungen.



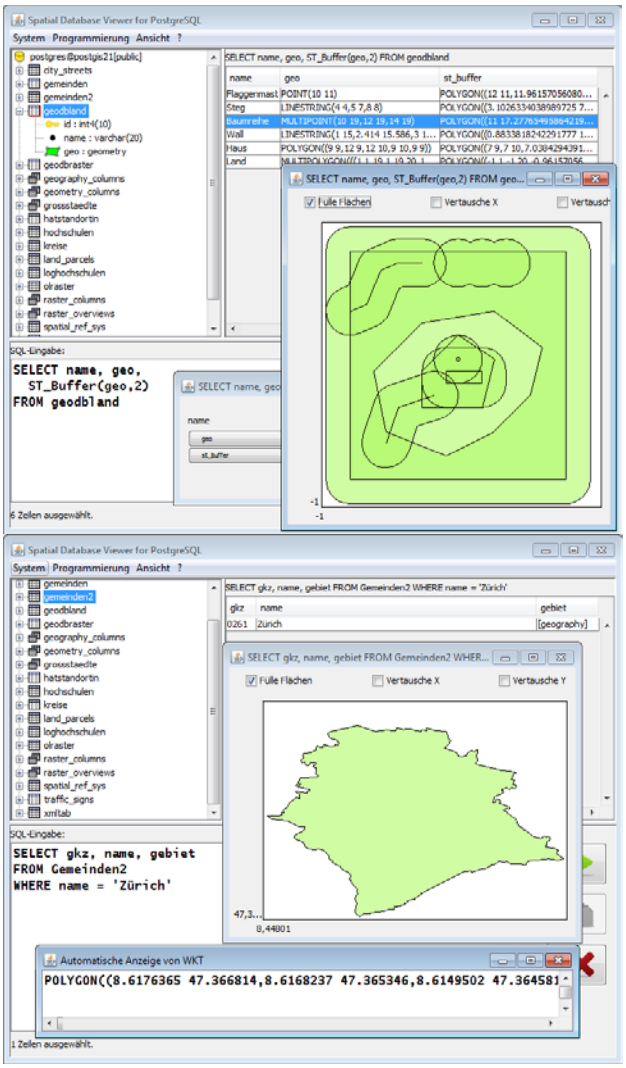


Abb. 3: Spatial Database Viewer für Oracle (obere Zeile) und PostgreSQL/PostGIS (untere Zeile)

Die Darstellungen illustrieren die textuelle und grafische Unterstützung von Geometrien mit Kreisbögen (Abb. 3, oben links), von Dreiecksvermaschungen (oben rechts), von geometrischen Funktionsresultaten (Pufferzonen unten links) und des `GEOGRAPHY`-Datentyps von PostGIS (unten rechts).

4 Zusammenfassung und Ausblick

Der Beitrag stellte eine konzeptionelle Erweiterung der Datenbankzugriffsschnittstelle JDBC für Geometriedatentypen vor, die einen einfachen Zugriff auf Geometrieobjekte gemäß dem Simple-Feature-Modell, auf Grafikobjekte (Java AWT) sowie auf standardisierte Textrepräsentationen (Well-Known Text) ermöglichen. Für die Geodatenbanksysteme Oracle Spatial und PostGIS/PostgreSQL wurde aufgezeigt, wie eine Implementierung über eine Erweiterung der vorhandenen Schnittstellen-Klassen erfolgen kann. Erprobt wurde der Ansatz an der Anwendung

„Spatial Database Viewer“, die die textuelle und grafische Anzeige unterschiedlicher Geometrietypen erlaubt.

Erforderlich ist eine Vervollständigung der vorgestellten Klassenstruktur und der Methoden: So sind bislang (mangels direkten Bedarfs) keine `CallableSpatialStatement`-Klassen implementiert worden. Auch wurden nicht alle dem Grundsatz nach möglichen `set-` und `update`-Methoden realisiert. Die Integration einer Konverter-Klasse nach Well-Known Binary ist ebenfalls angezeigt. Außerdem ist die Unterstützung des Geodatenbanksystems SpatiaLite geplant[10].

5 Literaturverzeichnis

BRINKHOFF, T., 2013: Geodatenbanksysteme in Theorie und Praxis, 3. Auflage. Wichmann, 524 Seiten.

CUSTER, A. (ed.), 2011: GeoAPI 3.0 Implementation Standard, Version: 3.0.0, OGC 09-083r3.

ISO, 2004: International Standard ISO 19125-1: 2004 – Geographic Information – Simple Feature Access – Part 1: Common Architecture.

ISO/IEC, 2011: International Standard ISO/IEC 13249-3:2011 – Information technology – Database languages – SQL Multimedia and Application Packages – Part 3: Spatial, 4th Edition.

MÜLLER, B. & WEHR H., 2012: Java Persistence API 2, Hanser, 339 Seiten.

ORACLE CORP., 2013: Oracle Spatial and Graph – Developer’s Guide.

POSTGIS, 2013: PostGIS 2.1.0 Manual.

[10] SpatiaLite Website: https://www.gaia-gis.it/fossil/libspatialite/index