

Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“
(UNIGIS MSc) am Zentrum für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

„Freie Datenbanksysteme zur Verwaltung, Analyse und Bearbeitung von Geodaten“ Räumliche Informationen in freien Datenbanken

vorgelegt von

Dipl.-Geol. Christian Schanz
U 1129, UNIGIS MSc Jahrgang 2004

Zur Erlangung des Grades
„Master of Science (Geographical Information Science & Systems) – MSc(GIS)“

Gutachter:
Ao. Univ. Prof. Dr. Josef Strobl

Karlsruhe, 15.02.2006

Erklärung

Ich versichere hiermit, die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel verfasst zu haben.

Karlsruhe, Februar 2006

.....

(Christian Schanz)

Zusammenfassung

In dieser Arbeit wird die Unterstützung zur Verwaltung von räumlichen Daten ausgewählter freier Datenbanksysteme verglichen. Hierfür werden zunächst die Grundlagen von freier Software und Open Source Software sowie von Datenbanksystemen dargestellt, bevor auf Standards und Besonderheiten zur Verwaltung von räumlichen Daten eingegangen wird. An ausgewählten Beispielen wird dann vorgestellt, inwieweit diese Anforderungen von Open Source Softwareprodukten erfüllt werden. Am Beispiel von PostgreSQL mit der Erweiterung PostGIS wird demonstriert, wie eine räumliche Datenbank angelegt und wie räumliche Abfragen durchgeführt werden können. Für ein Datenbanksystem wurde exemplarisch eine Anbindung an ein kommerzielles GIS in JAVA implementiert, um die prinzipielle Machbarkeit zu zeigen. Abschließend wird der Stand von freien Geodatenbanksystemen zusammengefasst und Trends der weiteren Entwicklung werden vorgestellt.

Stichwörter

Geodatenbanksysteme, räumliche Daten, Open Source, Java, PostgreSQL, PostGIS

Abstract

In this study the support for the management of spatial data for selected free database systems is evaluated. The basic principles of Free Software, Open Source Software and Database Systems are introduced. Subsequent the underlying standards as well as special characteristics of spatial data are discussed before the support of these requirements by selected Open Source products is presented. Then the Open Source database system PostgreSQL together with the spatial extension PostGIS are chosen to introduce how an spatial database is created and spatial queries are executed. Building on this a prototype database connection was implemented in Java. The study concludes with an overview of the current state of free geodatabase systems and Open Source and what changes one can expect over the several next years.

Keywords

Geodatabase-Systems, spatial data, Open Source, Java, PostgreSQL, PostGIS

Danksagung

Diese Master-These entstand im Rahmen des Universitätslehrganges „Geographical Information Science & Systems“ (UNIGIS MSc) am Zentrum für GeoInformatik (Z_GIS) der Paris Lodron-Universität Salzburg unter der Leitung von Ao.Univ. Prof. Dr. Josef Strobl. Bei ihm und dem übrigen UNIGIS-Team bedanke ich mich ganz herzlich für die zahlreichen interessanten Einblicke und Erkenntnisse, die mir dieser zweijährige Studiengang gebracht hat. Insbesondere das Modul Geo-Datenbank-Management hat mein Interesse an Geodatenbanken geweckt und mit zur Themenauswahl für diese Arbeit geführt.

Ein weiteres großes Dankeschön geht an die Mitarbeiterinnen und Mitarbeiter der Firma disy Informationssysteme GmbH in Karlsruhe. Hier sind insbesondere Marcus Briesen und Claus Hofmann zu nennen, die mir bei der Formulierung des Themas hilfreich zur Seite standen. In Markus Gebhard hatte ich immer einen Ansprechpartner, wenn Probleme mit der Java-Implementierung auftauchten. Auch dafür sage ich ein herzliches Dankeschön.

Nicht zu vergessen seien auch all die Open Source Entwickler, die die vorgestellten und verwendeten Open Source Projekte teilweise in ihrer freien Zeit entwickeln, der Allgemeinheit uneingeschränkt zur Verfügung stellen und darüber hinaus auch über die entsprechenden Mailinglisten bei Problemen unmittelbar ansprechbar sind. Ohne die Open Source Entwickler wäre diese Arbeit nicht möglich gewesen.

Inhaltsverzeichnis

1	Einführung	1
1.1	Motivation	1
1.2	Vorteile von Datenbanksystemen und die Integration von Fähigkeiten zur Verwaltung von Geodaten	4
2	Freie Software und Open Source Software	8
2.1	Was ist Freie Software – Was ist Open Source	8
2.2	Von Kathedralen und Basaren	9
2.3	Organisationen zur Förderung und Unterstützung freier Software	11
2.3.1	Free Software Foundation (FSF)	11
2.3.2	Open Source Initiative (OSI)	12
2.4	Open Source Lizenzen	14
2.4.1	GPL und LGPL	14
2.4.2	BSD-Lizenz	16
2.4.3	Weitere Lizenzen	17
3	Datenbanksysteme	18
3.1	Begriffsdefinitionen	18
3.2	Entwicklung der Datenbanksysteme	19
3.2.1	Relationale Datenbanksysteme	20
3.2.2	Objektrelationale Datenbanksysteme	22
3.3	Vergleich von (objekt-) relationalen Datenbanksystemen	23
4	Standards als Grundlagen für Geodatenbanksysteme	26
4.1	Das Open Geospatial Consortium	26
4.2	OGP/EPSSG	28
5	Indexmechanismen zur Verwaltung von Geodaten	30
5.1	Klassische Datenbankindizes	30

5.2	Räumliche Datenbankindizes	32
5.2.1	Der Quadtree	32
5.2.2	R-Baum	33
5.2.3	Suche in R-Bäumen	36
6	Freie Software zur Verwaltung von Geodaten	39
6.1	MySQL	39
6.1.1	Geschichte	40
6.1.2	Lizenzen	40
6.1.3	Eignung für Geodaten	41
6.2	PostgreSQL / PostGIS	41
6.2.1	Geschichte	42
6.2.2	Lizenzen	43
6.2.3	Eignung für Geodaten	44
6.3	JTS und GeoTools	44
6.4	Oracle Database 10g Express Edition	46
6.5	Unterstützung der Spezifikation „Simple Features for SQL“ durch Open Source Projekte	47
7	Verwaltung von Geodaten mit PostgreSQL und PostGIS	51
7.1	Anlegen einer Datenbank	51
7.2	Import von Geodaten	52
7.3	Generieren von Geodaten	54
7.4	Indizes in PostgreSQL und PostGIS	55
7.4.1	Indizes in PostgreSQL	56
7.4.2	Index in PostGIS	57
8	Datenbankkommunikation über JDBC am Beispiel von PostgreSQL/PostGIS	59
8.1	Die Schnittstelle zur Datenbank (JDBC)	59
8.2	Auslesen von Informationen über die Datenbankverbindung	61
8.3	Räumliche Ausdehnung und Auslesen der Geometrien innerhalb einer Bounding-Box	62
8.4	Auslesen der Nicht-Geometrie Daten für die ausgewählten Features	65
9	Implementierung einer PostgreSQL/PostGIS Datenbankanbindung	67
9.1	Implementierung	67
9.2	Beobachtungen und Weiterführung	70

Inhaltsverzeichnis

10 Fazit und Ausblick	73
Abkürzungsverzeichnis	76
Anhang A	78
Anhang B	84
Literaturverzeichnis	88

Abbildungsverzeichnis

1.1	Architekturen von Geoinformationssystemen	4
1.2	Verlagerung der GIS-Analyse-Aufgaben in die Geodatenbank	6
3.1	Aufbau eines Datenbanksystems)Verlagerung der GIS-Analyse-Aufgaben in die Geodatenbank	19
3.2	Beispiel einer Relationalen Datenbank	21
5.1	Darstellung eines B-Baumes	31
5.2	Darstellung eines Quadtree	33
5.3	Darstellung eines R-Baumes	35
5.4	Bounding-Boxen eines R-Baumes und die dazugehörigen Geometrieobjekte	36
5.5	Suche in einem R-Baum	38
7.1	Beispiele für die Visualisierung eines PostGIS-Index mit der PostgreSQL-Erweiterung „gevel“. Links: Darstellung eines synthetischen Datensatzes (verändert nach HOEFLE [20]), Rechts: Darstellung von Orten in Griechenland (verändert nach BARTUNOV UND SIGAEV [3]).	57
8.1	Schematische Darstellung des Zugriffs auf eine PostGIS Datenbank über die JDBC Schnittstelle.	60
8.2	Suchen von Features innerhalb eines vorgegebenen Extents	63
8.3	Zweistufige Geometrianfrage	65
9.1	Prototyp GISterm mit Zugriff auf eine PostgreSQL/PostGIS Datenbank . .	69
9.2	Schematische Darstellung des Zugriffs auf eine PostGIS Datenbank mit den geotools.	71

Tabellenverzeichnis

3.1	Vergleich von kommerziellen und Open Source Datenbanksystemen	24
6.1	Methoden und Exportformate von PostGIS, die über die OGC-Spezifikation hinausgehen (Auswahl)	43
6.2	Vergleich der Umsetzung der Simple Features For SQL für ausgewählte Implementierungen (I)	48
6.3	Vergleich der Umsetzung der Simple Features For SQL für ausgewählte Implementierungen (II)	50
7.1	Beispiel für die geometry_columns Tabelle einer PostgreSQL/PostGIS Datenbank mit verschiedenen Themen.	54

1 Einführung

1.1 Motivation

Aktuelle Geoinformationssysteme (GIS) sind das Ergebnis einer teils jahrzehntelangen Entwicklung. In diesem Zeitraum gab es graduelle Verbesserungen als auch mehrere revolutionäre Schritte, die stets auch der Entwicklung der allgemeinen Informatik und Computertechnik folgten (SCHILCHER, 1996; [52]).

Zunächst wurden Problemstellungen mit räumlichem Bezug vor allem als Forschungsprojekte bearbeitet. Das fand auf den damals nur in geringer Anzahl vorhandenen Großrechnern statt, deren Leistungsfähigkeit wesentlich geringer war als die heutiger Standard-Computer. Als erstes Geoinformationssystem der Welt gilt das Canada Geographic Information System (CGIS), das in den 60er Jahren des letzten Jahrhunderts in Kanada entwickelt wurde (BRINKHOFF, 2005 [6]).

Ausgehend von diesen mehr oder weniger experimentellen Einzelsystemen wurden schließlich Geographische Informationssysteme entwickelt, die auf Desktop-Workstations liefen und durch eine breitere Anwenderbasis bedienbar wurden. Diese Workstations waren aber noch nicht durch leistungsfähige Netzwerke verbunden und waren deshalb mehr oder weniger Dateninseln: Auf jedem einzelnen Rechner mussten die Daten vorgehalten werden. Daraus ergeben sich Probleme zum Beispiel im Bezug auf Datensynchronisation und Datensicherung.

Trotz der noch eingeschränkten Möglichkeiten des Datenhandlings entwickelten sich die Geographischen Informationssysteme nach und nach zu universell verwendbaren Werkzeugen, die heute auch auf Standard Computersystemen laufen, da insgesamt die Leistungsfähigkeit der Computer stark zugenommen hat. Entsprechend vielseitig gestalteten sich heute die Anwendungsmöglichkeiten der Systeme.

Parallel zu dieser Entwicklung erfolgte mit der zunehmenden Leistungsfähigkeit der Computer auch eine zunehmende Digitalisierung der allgemeinen Geschäftsprozesse. Das vielbeschworene „papierlose Büro“ ist zwar nach wie vor eine Zukunftsvision. Wichtig ist jedoch,

dass nicht mehr mit Zettelkästen und Schreibmaschine gearbeitet wird. Vielmehr liegen die Informationen heute in großen Teilen von Beginn an digital vor, und werden nur noch zur Weitergabe zusätzlich in Papierform ausgedruckt.

Für die Verwaltung dieser Daten werden heute in großen Umfang Datenbanksysteme verwendet. Durch die zentrale Datenhaltung lassen sich zum Beispiel die Probleme der Datensynchronisation und -sicherung lösen. Außerdem sind Datenbanksysteme mehrbenutzerfähig, so dass mehrere Anwender gleichzeitig auf Daten zugreifen und gegebenenfalls ändern können.

Die vergleichbare Verwaltung von Daten mit einem räumlichen Bezug (Geodaten) ist aus mehreren Gründen nicht so einfach. Geodaten besitzen thematische Eigenschaften (Sachdaten), die in konventionellen Datenbanksystemen problemlos gespeichert werden können. Darüber hinaus besitzen sie aber auch geometrische und topologische Eigenschaften, die in herkömmlichen Datenbanksystemen nur schwer abgebildet werden können (BRINKHOFF, 2005 [6]). Geodaten setzen sich also vereinfacht gesagt aus zwei Teilen zusammen, dem Geometrieteil, in dem die räumliche Ausdehnung verwaltet wird, sowie einem Nicht-Geometrieteil, in dem die Sachdaten zu einem Geometrieobjekt hinterlegt sind. Der Geometrieteil von Geodaten ist mit herkömmlichen Datenbanksystemen nicht ohne weiteres handhabbar.

Datenbanksysteme, die Geodaten verwalten können werden auch als Geodatenbank-Server oder Geodatenbanksysteme bezeichnet. Diese Arbeit beschränkt sich auf die Betrachtung von vektorbasierten räumlichen Daten. Georeferenzierte Rasterdaten lassen sich zwar prinzipiell auch mit Datenbanksystemen verwalten, jedoch ist dieser Anwendungsfall noch nicht so weit verbreitet und bedarf zum Teil völlig anderer Verwaltungsmechanismen.

Die Menge der Geodaten nimmt durch die Digitalisierung vorhandener Daten sowie durch die Nutzung neuer Techniken (GPS, location based services, ...) stark zu. Und mit der ständig wachsenden Datenmenge besteht mehr und mehr der Bedarf, diese auch adäquat in Datenbanksystemen zu verwalten. Der GIS-Report (BUHMANN UND WIESEL, 2003/2004; [7; 8]) beschreibt für den deutschsprachigen Raum einen „gesunden Zuwachs“ für den Umgang mit „verortbaren Informationen“. Speziell für den Bereich der Geodatenbank-Server wird für das Jahr 2003 ein Jahreszuwachs an damit verbundenen GIS-Arbeitsplätzen von +19% und für das Jahr 2004 von +38% angegeben.

Neben der technischen Entwicklung und zunehmenden Verbreitung von GIS Programmen und Geodatenbanksystemen ist aber auch eine neue Entwicklung in der Art, wie Software entwickelt wird, zu beobachten. Diese beruht zu einem großen Teil auf der heutzutage globa-

len Vernetzung, die auf die Öffnung des zunächst militärischen Internet (damals Milnet) für Universitäten und schließlich der Allgemeinheit zurückgeht. Durch neue Kommunikationsmöglichkeiten wie dem Internet ist es möglich geworden, dass Menschen mit gemeinsamen Interessen große Softwareprojekte entwickeln, ohne sich notwendigerweise jemals persönlich zu begegnen.

Das in der öffentlichen Wahrnehmung wahrscheinlich bekannteste Beispiel, wie erfolgreich ein solches kollaboratives Projekt sein kann, ist das Betriebssystem Linux. Dieses begann als persönliche „Programmierzugübung“ des finnischen Studenten Linus Torvald. Statt das Projekt jedoch alleine weiterzuentwickeln – oder einzustellen – stellte er den Quelltext der Allgemeinheit zur Verfügung und ermunterte andere, an der Weiterentwicklung oder der Fehlersuche mitzuarbeiten (TORVALDS UND DIAMOND, 2001 [61]).

Fairerweise muss dazugesagt werden, dass Linus Torvalds nicht der Erste war, der die Mechanismen einer gemeinsamen Entwicklung über das Internet nutzte. Allerdings ist sein Linux-Projekt eines der erfolgreichsten und vor allem bekanntesten Projekte dieser Art.

Durch die Veröffentlichung seines Quelltextes riskiert ein Entwickler, dass ein Konkurrent von seinen Ideen profitiert. Dies gilt vor allem im kommerziellen Umfeld, wo der Quelltext oft als Geschäftsgeheimnis gilt.

Andererseits ermöglicht die Veröffentlichung des Quelltextes, dass Menschen mit einem anderen Hintergrund ihre eigenen Ideen mit einbringen können, wenn sie sich an der Weiterentwicklung eines solchen Projekts beteiligen.

Um plumpen Ideenklau durch veröffentlichten Quelltext einzuschränken, wurden und werden verschiedene Lizenzmodelle entwickelt, die die Rechte des Entwicklers so weit schützen, wie dieser es wünscht.

Diese freien Entwicklungsprozesse sind mittlerweile etabliert und haben ihre Fähigkeit, komplexe und leistungsfähige Softwareprodukte hervorzubringen, bewiesen. Auch im Bereich der Datenbanktechnik wurden mehrere Datenbanksysteme mit solch offenen Entwicklungsprozessen entwickelt, die mittlerweile eine hohe Leistungsfähigkeit erreicht haben.

Und auch für die Verwaltung der in immer größerem Umfang anfallenden Geodaten entstehen Geodatenbanksysteme in solch offenen Entwicklungsprozessen.

1.2 Vorteile von Datenbanksystemen und die Integration von Fähigkeiten zur Verwaltung von Geodaten

Zu Beginn der Computerisierung wurden Daten in Form von Dateien in einem (lokalen) Dateisystem vorgehalten. In der heutigen Zeit werden Daten vielfach zentral in Datenbanken gehalten und können über Netzwerke auf vielen Client-Computern abgerufen werden.

Wie in der allgemeinen Computertechnik lässt sich dieser Wandel im Umgang mit Daten auch für Daten mit räumlichem Bezug (Geodaten) nachvollziehen. Auch hier wurden Daten mit Lagebezug zunächst lokal in Dateien abgespeichert. Diese dezentrale Speicherung bringt Probleme mit sich, sobald an mehreren Arbeitsplätzen mit den Daten gearbeitet werden soll. Dann liegt zunächst eine redundante Datenhaltung vor. Das bedeutet, dass dieselben Daten auf jedem Computer vorgehalten werden müssen. Wird mit diesen Daten gearbeitet, kann es zu Inkonsistenzen zwischen diesen unabhängig voneinander gespeicherten Daten kommen. Dies führt zu Problemen, wenn die Daten zusammengeführt werden sollen und eine konsistente Datensicherung ist ebenfalls nur schwer möglich (SCHEUGENPFLUG, 2005; [51]).

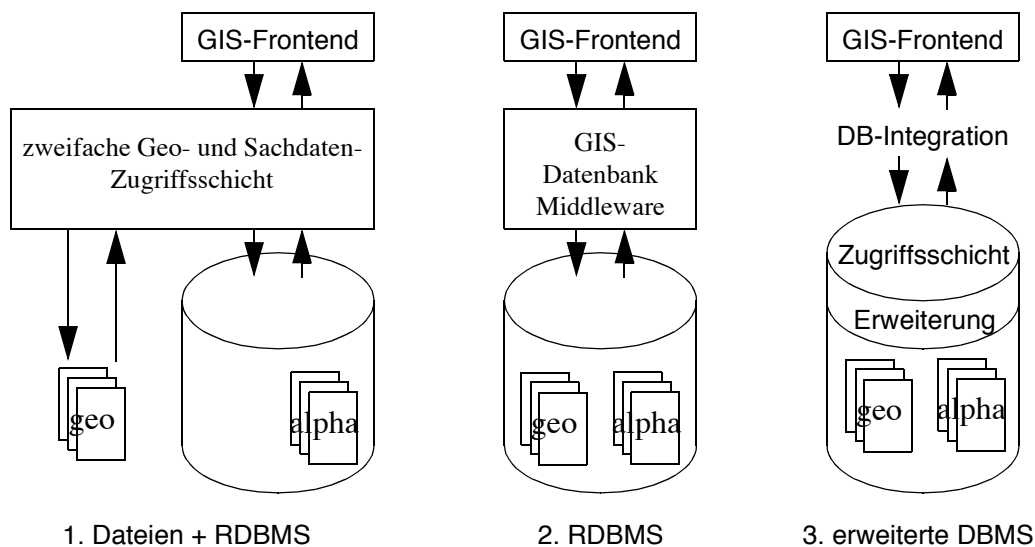


Abbildung 1.1: Architekturen von Geoinformationssystemen (Abbildung verändert nach Ziegler, 2002)

Architekturen von Geoinformationssystemen (Abbildung verändert nach ZIEGLER, 2002; [65]) In der allgemeinen Datentechnik wurden deshalb schon relativ früh Datenbanken für die Verwaltung von Daten verwendet. Die Entwicklung führte zu relationalen Daten-

banksystemen, die im Prinzip heute noch so verwendet werden. Neben einer konsistenten Datenhaltung, Mehrbenutzerfähigkeit sowie einer möglichen zentralen Datenhaltung bieten Datenbanksysteme über Indizes die Möglichkeit, schnell auf bestimmte Datensätze zuzugreifen zu können. Ein solcher Index ist vereinfacht gesagt vergleichbar mit einem Zahlenstrahl, auf dem die Position jedes Datensatzes für ein bestimmtes Suchkriterium hinterlegt ist. Dies führt zu einer Ordnung, die eine einfache Suche erlaubt.

Räumliche Daten lassen sich über einen solchen einfachen Index nicht erfassen, da ein Punkt über Koordinaten in mindestens zwei Dimensionen definiert wird und sich nicht eindeutig auf einem zahlenstrahlartigen Index abbilden lässt. Geometrien (Linien, Polygone etc.), die aus mehreren Punkten zusammengesetzt sind, verkomplizieren die Indizierung von räumlichen Daten weiter.

Deshalb haben Datenbanksysteme nur schrittweise Einzug in die GIS-Welt gehalten. In Abbildung 1.2 sind verschiedene Stadien dieser zeitlichen und technischen Entwicklung dargestellt. Zunächst wurden die Geodaten aufgeteilt: Die reinen Geometrieinformationen wurden weiterhin lokal in Dateien verwaltet, während die dazugehörigen Sachinformationen in Datenbanken abgelegt wurden. Eine Zwischenschicht (engl. Middleware) muss hierbei sicherstellen, dass das GIS-Frontend, also das Programm, mit dem der Anwender interagiert, über eine einheitliche Schnittstelle transparenten Zugriff sowohl auf die Geometrieinformationen als auch auf die Sachinformationen hat (ZIEGLER, 2002; [65]).

Um diese Trennung und gesonderte Behandlung der Daten zu vermeiden, wurden Wege gesucht, um die Geometrieinformationen ebenfalls mit Hilfe eines (relationalen) Datenbanksystems abzubilden. Da die Datenbanksysteme noch nicht in der Lage waren, die Geometrien „zu verstehen“, also als eigenständige Objekte zu erfassen, wurden die Geometrieinformationen als Binary Large Objects (BLOBs) gekapselt. BLOBs sind ein Datentyp, mit dem quasi beliebige Informationen (Töne, Bilder, etc.) in einer Datenbank abgelegt werden können. Eine Indizierung von BLOBs und damit ein gezielter schneller Zugriff ist jedoch nicht unbedingt möglich. Bei diesem Ansatz befinden sich zwar die Geometrie- und Sachdaten gemeinsam in einer Datenbank. Damit das GIS-Frontend mit diesen Daten arbeiten kann, ist aber immer noch eine Zwischenschicht erforderlich, die die Auswertung der BLOBs übernimmt. Ein bekannter Vertreter einer solchen Middleware ist zum Beispiel ArcSDE von ESRI.

Dieser Ansatz ist auch heute noch weit verbreitet. Er bietet schon viele Vorteile, die die Verwendung eines Datenbanksystems mit sich bringt, wie zum Beispiel gemeinsamen Zugriff und zentrale Datensicherung.

Der Trend geht jedoch hin zu Datenbanksystemen, die Geometrien tatsächlich als solche verwalten können. Außerdem wurden Standards etabliert, die es einem GIS-Frontend im Prinzip erlaubt, unmittelbar auf einer Datenbank zu arbeiten. Diese Entwicklungsstufe beruht auf Erweiterungen der Datenbanksysteme. Geometrien können damit als Objekte abgelegt werden. Somit hat das Datenbanksystem einen „Überblick“ über die hinterlegten Geometrien und kann über geeignete Indizierungsmechanismen gezielte und schnelle Suche von räumlichen Informationen durchführen. Darüber hinaus können schon durch das Datenbanksystem geometrische Operationen und Analysen wie zum Beispiel das Verschneiden von Geometrien bewältigt werden. Abbildung 1.2 zeigt, wie die GIS-Analyseaufgaben, hier dargestellt durch eine Glühbirne, ursprünglich von dem GIS-Frontend wahrgenommen wurden. Im nächsten Schritt der Abstrahierung übernimmt eine Middleware, wie zum Beispiel ArcSDE von ESRI, diese Analyseaufgaben. Verschiedene Datenbanksysteme sind quasi als dritte Abstraktionsstufe heute schon in der Lage, diese Analyseaufgaben schon selber oder durch Erweiterungen durchzuführen. Oracle stellt dazu zum Beispiel die Spatial Extension zur Verfügung, das Open Source Datenbanksystem PostgreSQL lässt sich entsprechend durch die Einbindung PostGIS erweitern (ZIEGLER, 2002; [65]).

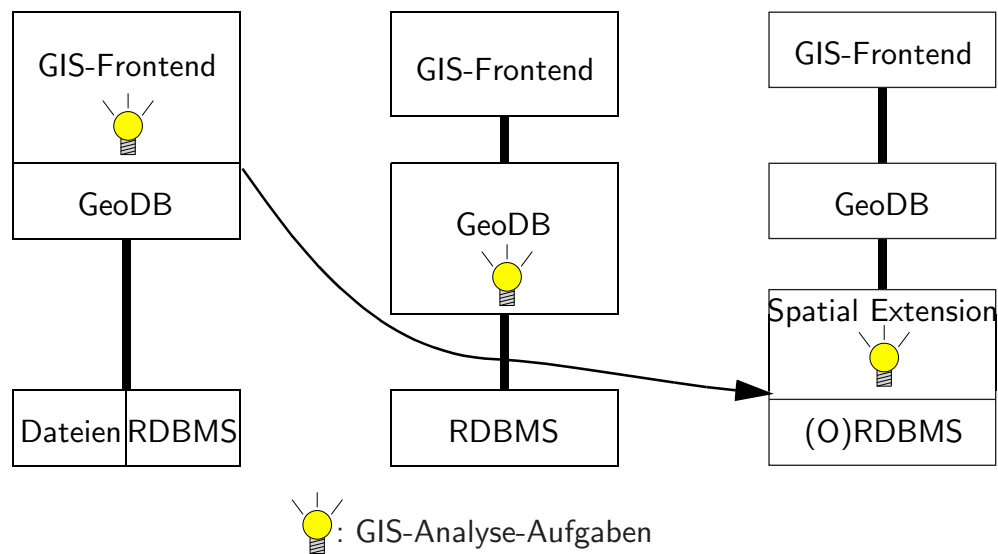


Abbildung 1.2: Verlagerung der GIS-Analyse-Aufgaben vom GIS-Frontend zur Geodatenbank (Abbildung verändert nach Ziegler, 2002)

Zu beobachten ist also, dass sich die GIS-Analyse-Aufgaben weg von den Anwendungsprogrammen hin zu den Datenbanksystemen verschieben und damit universell verwendet werden können.

Allerdings gehen die Fähigkeiten moderner GIS-Programme über die Fähigkeiten der Da-

tenbanksysteme hinaus, so dass viele Arbeitsschritte nach wie vor im GIS-Frontend stattfinden.

2 Freie Software und Open Source Software

In zunehmendem Maße tauchen auch in der öffentlichen Berichterstattung Begriffe wie „Freie Software“ und „Open Source“ auf. Nicht immer werden die Begriffe jedoch im richtigen Kontext verwendet. Deshalb werden im Folgenden diese Begriffe mit ihren Bedeutungen vorgestellt und wesentliche Unterschiede gezeigt.

Eng mit den Begrifflichkeiten verbunden sind verschiedene Philosophien, wie Softwareprojekte entwickelt und organisiert werden. Die gegensätzlichen Ideen von geschlossenen und offenen Entwicklungsprozessen mit wichtigen Auswirkungen auch für den Anwender werden erläutert.

Bedeutend für die Koordinierung und Förderung von freier Software sind verschiedene gemeinnützige Organisationen. Einige wichtige Organisationen werden zusammen mit ihren Philosophien vorgestellt. Diese Organisationen betreuen auch Lizenzmodelle für die Veröffentlichung und Benutzung von freier Software. Für die beiden wichtigsten, die GPL und die BSD-Lizenz, werden die Nutzungsbestimmungen vorgestellt, die für den Quelltext sowie für ausführbare Programme gelten.

2.1 Was ist Freie Software – Was ist Open Source

Der Begriff „Freie Software“ (englisch: Free Software) hat sowohl im Deutschen als auch im Englischen zwei wesentliche Bedeutungen. Einerseits bezeichnet er Software, die dem Benutzer gratis zur Verfügung steht. Der Begriff „Frei“ kann aber andererseits auch die Bedeutung von „Freiheit“ (freie Meinungsäußerung) haben.

Während die erste Bedeutung rein finanzielle Aspekte abdeckt, ist die Zweite weitreichender: Sie drückt aus, dass dem Benutzer vom Entwickler Freiheiten zugestanden werden, was er mit einem Programm tun darf. Diese Freiheiten gehen weit über die eigentliche Anwendung des Programmes hinaus. Die Freiheit eines Programme drückt sich nach diesem

Begriffsverständnis in der Form aus, dass der dem Programm zugrundeliegenden Quelltext von dem oder den Programmierern veröffentlicht wird. Jeder Interessierte kann daher Einblick nehmen und bei Bedarf sogar selber Änderungen am Quelltext vornehmen. Das kann zum Beispiel interessant sein um Fehler auszumerzen, Funktionen für den eigenen Gebrauch zu implementieren oder auch, um ein Programm auf eine komplett andere Plattform zu portieren. Außerdem ist eine gewisse Zukunftssicherheit gewährleistet. Falls der ursprüngliche Entwickler kein Interesse an oder keine Zeit mehr für eine Weiterentwicklung hat, kann es von anderen, für die das Programm noch wichtig ist, weiter betreut werden (RENNER ET AL., 2005; [49]).

Der Begriff der „Free Software“ wurde und wird vor allem von der „Free Software Foundation (FSF)“ geprägt. Diese gemeinnützige Organisation wurde im Jahr 1985 von Richard Stallman gegründet, der auch Hauptinitiator der GPL, einer der wichtigsten Open Source Lizenzen, ist. Auf die Bedeutung und Besonderheiten dieser Lizenz wird in Kapitel 2.4.1 eingegangen.

Da die Doppelbedeutung des Wortes „Frei“ beziehungsweise „Free“ für viele unbefriedigend und nicht eindeutig genug war, wurde etwa ab dem Jahr 1998 vor allem von Eric (E. S.) Raymond, Bruce Perens und Tim O’Reilly der Begriff „Open Source“ etabliert. Eric Raymond und Bruce Perens gründeten im Jahr 1998 die „Open Source Initiative (OSI)“. Die OSI ist eine gemeinnützige Organisation zur Förderung von Open Source Software. Tim O’Reilly ist Gründer und Chef des O’Reilly-Verlages für Computerbücher, ist daneben aber auch als Veranstalter von Konferenzen und Fürsprecher für Open Source Software bekannt.

In dieser Arbeit wird der Begriff „Freie Software“ im Sinne von Freiheit des Quelltextes, also als „Open Source“ verwendet, sofern nichts anderes vermerkt ist.

2.2 Von Kathedralen und Basaren

Den Gegensatz zur „Open Source“ Software bildet die so genannte „Closed Source“ Software, die oft auch als proprietäre oder kommerzielle Software bezeichnet wird (wenn diese Begriffe auch nicht exakt dasselbe beschreiben). Darunter versteht man Software, die nur in Form von ausführbaren Dateien weitergegeben wird. Der Quelltext steht nur einem eingeschränkten Personenkreis, meistens der Firma, die das Programm entwickelt, zur Verfügung.

Die gegensätzlichen Entwicklungsmodelle wurden im Jahr 1997 von Eric Raymond, dem Mitbegründer der Open Source Initiative (OSI), erstmals als Kathedralen-Modell und

Basar-Modell bezeichnet (RAYMOND, 2002; [14]) .

Die Entwicklung von „Closed Source“ Software vergleicht Raymond mit dem Bau einer Kathedrale. Ein Baumeister hat Einsicht in die Pläne und sorgt für die Fertigstellung des Bauwerkes. Der Anwender entspricht dem Besucher der fertigen Kathedrale und hat auf die Gestaltung und Fertigstellung des Gebäudes (= Programm) keinen großen Einfluss.

Dagegen soll der Vergleich von „Open Source“ Software mit einem Basar deutlich machen, dass viele oft auch gleichrangige „Händler“ handeln und feilschen, bis ein Ergebnis zustande kommt. Die Entwicklung erfolgt meist in offener Diskussion, zum Beispiel über Mailinglisten. Liegen unterschiedliche Meinungen zu oder Lösungsvorschläge für ein bestimmtes Problem vor kann es vorkommen, dass mehrere alternative Varianten implementiert werden und sich später eine Version durchsetzt. Außerdem werden erfolgreiche Projekte gerne in anderen Projekte mit verwendet.

Als ein Beispiel für diese modulare Wiederverwendung sei hier die in dieser Arbeit zu einem späteren Punkt näher vorgestellte Datenbankerweiterung PostGIS genannt, die das Datenbanksystem PostgreSQL um Geodatenfunktionen erweitert. Über die Open Source Bibliothek GEOS bezieht PostGIS viele standardisierte Funktionen zum Umgang mit georeferenzierten Daten. Und GEOS wiederum beruht auf dem Open Source Projekt JTS. Diese Zusammenstellung soll an dieser Stelle nur verdeutlichen, dass es zwischen vielen Open Source Projekte enge Verflechtungen gibt. In Kapitel 6 wird auf die Verbindungen dieser Open Source Projekte näher eingegangen. Durch die enge Verbindung dieser Projekte wird erkennbar, dass sich das Endprodukt – ein Datenbanksystem, das Geodaten verwalten kann – aus dem Zusammenspiel von vielen separat entwickelten Open Source Projekten ergibt. Diese Modularität gilt als ein großer Vorteil der freien Entwicklungsmodelle.

Darüber, welches das „bessere“ Entwicklungsmodell ist, wird teilweise mit fast religiösem Eifer gestritten. Eine Bewertung solcher Art ist aber nicht Aufgabe dieser Arbeit. Die Erfahrung zeigt, dass sowohl nach dem Kathedralen- als auch nach dem Basar-Modell komplexe Softwareprojekte realisiert werden können.

Beide Entwicklungsmodelle koexistieren heute nebeneinander und nehmen teilweise auch Anleihen bei dem konkurrierenden Modell. Viele Open Source Projekte werden zum Beispiel überwiegend von einer Firma betreut, während andererseits zum Beispiel Microsoft als wahrscheinlich bekanntester Vertreter des Kathedralen-Modells mit seiner Shared Source Initiative (MICROSOFT, 2004; [30]) auch Techniken der Open Source Entwicklung evaluiert. Andere Firmen, wie zum Beispiel IBM (z. B. Eclipse) und SUN (z. B. Openoffice.org), stellen einzelne, ursprünglich kommerziell entwickelte Programme, unter freie Softwareli-

zenzen. Statt Einnahmen aus Softwarelizenzen wollen sie zum Beispiel als Dienstleister an Beratungs- und Wartungsverträgen oder kundenspezifischen Anpassungen partizipieren.

2.3 Organisationen zur Förderung und Unterstützung freier Software

Mittlerweile gibt es zahlreiche Organisationen, deren Ziel die Förderung von freier Software ist. Viele beschränken sich auf einzelne oder mehrere Projekte, zum Beispiel die Eclipse Foundation ([60]) oder die Apache Software Foundation ([59]).

Daneben gibt es aber auch Organisationen, die sich der Idee von freier Software allgemein annehmen. Zwei wichtige Organisationen sind die Free Software Foundation (FSF) und die Open Source Initiative (OSI).

2.3.1 Free Software Foundation (FSF)

Die Free Software Foundation (FSF) wurde im Jahr 1985 von Richard Stallmann, Robert Chassel und anderen gegründet. Sie entstand aus dem Umfeld des GNU Projektes (eine Abkürzung für GNU is Not UNIX), dessen erstes Ziel die Schaffung eines freien UNIX-ähnlichen Betriebssystems war (WILLIAMS, 2002; [64]).

Die Organisation vertritt die Auffassung, dass der Begriff „frei“ nicht bloß „kostenlos“ bedeutet, sondern im Sinne von „Freiheit“ zu verwenden ist. So soll jeder freien Quelltext verwenden dürfen, muss aber dann seine Änderungen wieder der Öffentlichkeit zur Verfügung stellen.

Die Free Software Foundation ist federführend bei der Weiterentwicklung von zwei Softwarelizenzen (GPL und LGPL), die diese Anforderungen umsetzen. Diese beiden Lizenzen werden in Abschnitt 2.4.1 vorgestellt.

Die Hauptaufgaben der Free Software Foundation liegen in der Öffentlichkeitsarbeit und im Sammeln von Spenden, die für die Entwicklung von freier Software oder Rechtsstreitigkeiten verwendet werden. Nach wie vor gibt es enge Beziehungen zum GNU-Projekt.

Die Philosophie hinter der Organisation sowie den Lizenzen ist in ausführlich in STALLMAN (2002; [55]) dargestellt.

2.3.2 Open Source Initiative (OSI)

Im Gegensatz zur Free Software Foundation vertritt die Open Software Initiative (OSI) einen weniger ideologischen Standpunkt, was die Verwendung von quelltextoffener Software angeht. Die Initiative wurde im Jahr 1998 gegründet, unter anderem als eine Reaktion auf die Freigabe des Quelltextes des Internet-Browsers „Netscape Navigator“ durch die Firma Netscape.

Mit dem Begriff „Open Source“ sollte ein Begriff geprägt werden, der insbesondere im Geschäftsbereich eingängiger und weniger abschreckend als der Begriff „Free Software“ sein sollte. Zu den Gründungsmitgliedern gehörte unter anderem Eric S. Raymond von dem im Kapitel 2.2 bereits als Autor des Artikels „The Cathedral and the Bazaar“ die Rede war (OPEN SOURCE INITIATIVE; [39]).

Die Open Source Initiative hat eine Definition für den Begriff „Open Source“ veröffentlicht, die zehn Punkte umfasst (OPEN SOURCE INITIATIVE; [42]) und aus der Definition der freien Linux-Distribution Debian hervorgegangen ist. Diese zehn Punkte müssen durch eine Lizenz erfüllt werden, damit sie sich als „Open Source“-Lizenz bezeichnen darf. Die folgende deutsche Übersetzung wurde weitgehend von RONNEBURG; [50]) übernommen.

1. **Freie Weitergabe** (Free Redistribution)

Die Lizenz darf niemanden in seinem Recht einschränken, die Software als Teil eines Software-Paketes, das Programme unterschiedlichen Ursprungs enthält, zu verschenken oder zu verkaufen. Die Lizenz darf für den Fall eines solchen Verkaufs keine Lizenz- oder sonstigen Gebühren festschreiben.

2. **Quellcode** (Source Code)

Das Programm muss den Quellcode beinhalten. Die Weitergabe muss sowohl für den Quellcode als auch für die kompilierte Form zulässig sein. Wenn das Programm in irgendeiner Form ohne Quellcode weitergegeben wird, so muss es eine allgemein bekannte Möglichkeit geben, den Quellcode zum Selbstkostenpreis zu bekommen, vorzugsweise als gebührenfreien Download aus dem Internet. Der Quellcode soll die Form eines Programms sein, die ein Programmierer vorzugsweise bearbeitet. Absichtlich unverständlich geschriebener Quellcode ist daher nicht zulässig. Zwischenformen des Codes, so wie sie etwa ein Präprozessor oder ein Konverter erzeugt, sind unzulässig.

3. **Abgeleitete Software** (Derived Works)

Die Lizenz muss Veränderungen und Derivate zulassen. Außerdem muss sie es zulas-

sen, dass die solcherart entstandenen Programme unter denselben Lizenzbestimmungen weitervertrieben werden können wie die Ausgangssoftware.

4. **Unversehrtheit des Quellcodes des Autors** (Integrity of The Author's Source Code)
Die Lizenz darf die Möglichkeit, den Quellcode in veränderter Form weiterzugeben, nur dann einschränken, wenn sie vorsieht, dass zusammen mit dem Quellcode so genannte „Patch files“ weitergegeben werden dürfen, die den Programmcode bei der Kompilierung verändern. Die Lizenz muss die Weitergabe von Software, die aus verändertem Quellcode entstanden ist, ausdrücklich erlauben. Die Lizenz kann verlangen, dass die abgeleiteten Programme einen anderen Namen oder eine andere Versionsnummer als die Ausgangssoftware tragen.
5. **Keine Diskriminierung von Personen oder Gruppen** (No Discrimination Against Persons or Groups)
Die Lizenz darf niemanden benachteiligen.
6. **Keine Einschränkungen bezüglich des Einsatzfeldes** (No Discrimination Against Fields of Endeavor)
Die Lizenz darf niemanden daran hindern, das Programm in einem bestimmten Bereich einzusetzen. Beispielsweise darf sie den Einsatz des Programms in einem Geschäft oder in der Genforschung nicht ausschließen.
7. **Weitergabe der Lizenz** (Distribution of License)
Die Rechte an einem Programm müssen auf alle Personen übergehen, die diese Software erhalten, ohne dass für diese die Notwendigkeit bestünde, eine eigene, zusätzliche Lizenz zu erwerben.
8. **Die Lizenz darf nicht auf ein bestimmtes Produktpaket beschränkt sein** (License Must Not Be Specific to a Product)
Die Rechte an dem Programm dürfen nicht davon abhängig sein, ob das Programm Teil eines bestimmten Software-Paketes ist. Wenn das Programm aus dem Paket herausgenommen und im Rahmen der zu diesem Programm gehörenden Lizenz benutzt oder weitergegeben wird, so sollen alle Personen, die dieses Programm dann erhalten, alle Rechte daran haben, die auch in Verbindung mit dem ursprünglichen Software-Paket gewährt wurden.
9. **Die Lizenz darf die Weitergabe zusammen mit anderer Software nicht einschränken** (License Must Not Restrict Other Software)

Die Lizenz darf keine Einschränkungen enthalten bezüglich anderer Software, die zusammen mit der lizenzierten Software weitergegeben wird. So darf die Lizenz z. B. nicht verlangen, dass alle anderen Programme, die auf dem gleichen Medium weitergegeben werden, auch quelloffen sein müssen.

10. **Die Lizenz muss Technologie-Neutral sein** (License Must Be Technology-Neutral)
Keine Bestimmung der Lizenz darf sich auf eine bestimmte Technologie oder Bedienoberfläche gründen.

2.4 Open Source Lizenzen

Über Lizenzen werden Nutzungsrechte für Softwareprodukte geregelt. Für „Closed Source“ Programme wird über Lizenzen (auch EULA, End User Licence Agreement) zum Beispiel geregelt, auf wie vielen Computern die Software installiert werden darf. Die Lizenz bezieht sich dann auf das ausführbare Programm und untersagt in der Regel auch das Dekodieren (reverse engineering) der Software.

Für „Open Source“ Programme bezieht sich die Lizenz dagegen vor allem, aber nicht ausschließlich, auf den zugrundeliegenden Quelltext. Durch die Lizenz wird geregelt, ob der Quelltext angepasst oder in eigenen Produkten verwendet werden darf, ob Änderungen am Quelltext wiederum veröffentlicht werden müssen etc.. Eine Lizenz kann aber zum Beispiel auch bestimmen, dass der Autor bei regelmäßiger Nutzung der Software vom Anwender eine Postkarte erhalten möchte.

Durch Lizenzen wird das Urheberrecht nicht berührt. Das heisst, dass auch für Quelltext, der durch die Lizenz ohne Einschränkungen verwendet werden darf, nach wie vor die Urheberrechte beim Autor liegen. Der Autor wiederum hat das Recht, seinen Quelltext unter verschiedenen Lizenzen für verschiedene Zwecke anzubieten. Ein Beispiel hierfür wird in Kapitel 6.1 vorgestellt.

Der Begriff des Copyright stammt aus dem angloamerikanischen Rechtssystem und entspricht in etwa dem Urheberrecht. Aufgrund der unterschiedlichen Rechtssysteme lassen sich die Begriffe jedoch nicht gleichsetzen.

2.4.1 GPL und LGPL

Die GNU GPL (GNU General Public License) ist eine der bekanntesten und am weitesten verbreiteten Open Source Lizenzen. Üblicherweise wird sie verkürzt nur als GPL bezeichnet.

net. Sie wurde von Richard Stallmann zusammen mit dem Rechtsprofessor Eben Moglen entworfen und erstmals im Jahr 1989 veröffentlicht. Die Lizenz wird heute von der Free Software Foundation (FSF) herausgegeben. Der Hintergrund, der zur Gründung des GNU Projektes und daraus folgend der GPL führte ist in STALLMAN (2001; [56]) dargestellt.

Zur Zeit ist die Lizenz in der Version 2 aus dem Jahr 1991 aktuell (STALLMAN UND MOGLEN; [57]). Die Version 3 der Lizenz ist in Vorbereitung und soll vor allem besser an international unterschiedliche Rechtssysteme sowie an aktuelle technische und gesellschafts-politische Aspekte angepasst sein (Softwarepatente, Digitale Rechteverwaltung, ...). Die Grundzüge und -prinzipien der bisherigen GPL sollen darin aber weitergeführt werden (JAEGER UND METZGER, 2006; [27]). Die GPL in der Version 3 soll voraussichtlich noch im Jahr 2006 fertiggestellt und veröffentlicht werden. Die neuen Ideen werden zum Teil kontrovers diskutiert, so dass ein sicheres Veröffentlichungsdatum derzeit nicht feststeht ([16]).

Die zur Zeit aktuelle GPL (Version 2) besagt, dass unter der GPL stehender Quelltext für eigene Projekte verwendet und verändert werden darf. Sobald Binärdateien, die Quelltext unter der GPL enthalten, veröffentlicht werden – also nicht nur für den Eigenbedarf verwendet werden – muss sämtlicher davon abgeleiteter Quellcode ebenfalls unter der GPL veröffentlicht werden.

Microsoft bezeichnet die GPL deshalb als „viral“ (MICROSOFT; [(30)]), da eigener Quelltext, der Quelltext unter der GPL-Lizenz verwendet, „infiziert“ wird und ebenfalls unter der GPL veröffentlicht werden muss.

Um einem Großteil dieser Bedenken zu begegnen, wurde neben der GPL zusätzlich die LGPL (Lesser General Public License) eingeführt. Die LGPL ist eine besondere Lizenz für Programmbibliotheken, die von anderen Programmen verwendet werden können, ohne in das eigene Programm integriert werden zu müssen. Solange man in seinem Programm Bibliotheken, die unter der LGPL stehen, nur aufruft, muss man den Quelltext seines eigenen Programmes nicht offenlegen. Lediglich Änderungen, die man eventuell an der Bibliothek selber durchführt muss man vergleichbar wie bei der GPL veröffentlichen, wenn das Softwareprodukt nicht nur für den eigenen Gebrauch entwickelt wird.

Die GPL baut auf dem Prinzip des „Copyleft“ auf. Der Begriff Copyleft geht auf die homonyme Bedeutung des englischen Begriffes „right“ zurück, der sowohl „das Recht“ als auch „rechts“ (Richtung) bedeutet. Copyleft bedeutet, dass für ein Programm Copyrightansprüche geltend gemacht werden. Durch die Lizenz werden allerdings dem Benutzer weitgehende Rechte (Verwendung, Abänderung und Weiterverbreitung) eingeräumt, unter den Bedin-

gungen, die die GPL beschreibt. Für Copyleft wird analog zum Copyrightsymbol (©) gelegentlich das Symbol Ⓒ verwendet.

In diesem Sinne gewährt die GPL jedem die folgenden vier Freiheiten als Bestandteil der Lizenz (Deutsche Übersetzung: WIKIPEDIA; [63], Englisch Original: [57]):

1. Ein Programm für jeden Zweck zu nutzen (und nicht durch Lizenzen eingeschränkt zu sein)
2. Kopien des Programms kostenlos zu verteilen (wobei der Quellcode mitverteilt oder dem Empfänger des Programms auf Anfrage zur Verfügung gestellt werden muss)
3. Die Arbeitsweise eines Programmes zu studieren und es den eigenen Bedürfnissen entsprechend zu ändern (die Verfügbarkeit des Quellcodes ist Voraussetzung dafür)
4. Auch nach 3. veränderte Versionen des Programms unter den Regeln von 2. vertreiben zu dürfen (wobei der Quellcode wiederum mitverteilt oder dem Empfänger des Programms auf Anfrage zur Verfügung gestellt werden muss)

Wie eingangs bereits erwähnt ist die GPL eine der am meisten verwendeten Open Source Lizenzen. Das wahrscheinlich wichtigste Projekt, welches diese Lizenz verwendet ist vermutlich der Linux-Kernel. Die GPL wurde auch schon mehrfach erfolgreich vor Gericht gegen Firmen durchgesetzt, die zwar GPL-Quelltext für eigene Projekte verwendeten, aber diese Änderungen nicht wiederum veröffentlichten (JAEGER UND METZGER, 2006; [27]).

2.4.2 BSD-Lizenz

Die Berkeley-Software-Distribution (BSD) Lizenz ist eine weitere weit verbreitete Open Source Lizenz. Sie ist wesentlich weniger restriktiv als die GPL, was den Umgang mit dem Quelltext angeht und infolgedessen ist auch der Lizenztext wesentlich kürzer.

Der Lizenztext besteht in der aktuellen Version (1999) aus lediglich drei Absätzen, die den Umgang mit dem Quelltext regeln (OPEN SOURCE INITIATIVE, 1999;[41]). Verkürzt besagen sie das folgende:

1. Weitergegebener Quelltext muss die Lizenzbedingungen enthalten
2. Weitergegebenen Programmen muss eine Kopie der Lizenzbedingungen beiliegen

3. Der Name des Lizenzgebers darf ohne schriftliche Genehmigung nicht für Werbezwecke verwendet werden.

Die BSD-Lizenz geht auf Softwareentwicklungen an der Universität von Kalifornien (Berkeley) in den 70er Jahren des letzten Jahrhunderts zurück. Da die Software mit Hilfe von öffentlichen Geldern entwickelt wurde, sollte sie auch möglichst ohne Beschränkungen wieder der Allgemeinheit zur Verfügung stehen. Die Berkeley Software Distribution war schließlich ein vollständiges UNIX-System. Daraus sind die heutigen UNIX-Systeme FreeBSD, NetBSD und OpenBSD hervorgegangen.

Im Gegensatz zur GPL, die an denjenigen, der den Quelltext verwendet auch gewisse Anforderungen stellt, ist die BSD-Lizenz liberaler formuliert. Sie schreibt lediglich das Copyright fest. Jedem ist es dann mehr oder weniger freigestellt, den Quelltext zu verändern und weiterzuverbreiten. Der Entwickler darf Quelltext, der der BSD-Lizenz unterliegt auch in Closed-Source Programme einbinden und wird nicht dazu gezwungen dann seinen eigenen Code wieder offenzulegen. Es reicht aus, dem verbreiteten Programm eine Kopie der Lizenz beizufügen. Die Lizenz ist damit nach der Definition von Microsoft nicht „viral“ (MICROSOFT; [30]).

2.4.3 Weitere Lizenzen

Neben den beiden vorgestellten und am weitesten verbreiteten Lizenzen zur Weitergabe von Open Source Software gibt es unzählige weitere. Sie alle dienen dazu, den Quelltext für ein Softwareprodukt anderen zugänglich zu machen. Dabei gibt es allerdings unterschiedliche Vorstellungen, wer mit diesem Sourcecode unter welchen Bedingungen arbeiten darf, ob eigene Änderungen ebenfalls veröffentlicht werden müssen und ob resultierende Programme kommerziell verwertet werden dürfen.

Zu Problemen kann es kommen, wenn Teile unterschiedlicher OpenSource-Programme in einem Projekt gemeinsam genutzt werden sollen. Dann müssen die Lizenzmodelle der einzelnen Programme untereinander kompatibel sein. Einen Vergleich verschiedener Lizenzmodelle sowie eine Bewertung ihrer Kompatibilität zeigen FREE SOFTWARE FOUNDATION [15] oder OPEN SOURCE INITIATIVE [40].

3 Datenbanksysteme

Neben dem Begriff der freien Software beziehungsweise der Idee von Open Source Software bilden Datenbanksysteme beziehungsweise die damit verwalteten Datenbanken den zweiten Schwerpunkt dieser Arbeit.

Zur Einführung werden zunächst grundlegende Begriffe und Konzepte vorgestellt. Für ausgewählte gängige kommerzielle und Open Source Datenbanken werden die Fähigkeiten zur Verwaltung von alphanumerischen Daten sowie der weitere Funktionsumfang zusammengefasst dargestellt. Diese Eigenschaften bilden die Basis für geometrische Erweiterungen, die zur Verwaltung von räumlichen Daten erforderlich sind und in den folgenden Kapiteln behandelt werden.

Da Geodaten neben der räumlichen Komponente auch alphanumerische Sachdaten umfassen, sind der Funktionsumfang und die Leistungsfähigkeit eines Datenbanksystems zur Verwaltung solcher nicht-räumlichen Daten wichtige Faktoren für die Evaluierung, welches System für ein bestimmtes Einsatzszenario geeignet ist.

3.1 Begriffsdefinitionen

Der Begriff „Datenbank“ wird uneinheitlich verwendet. So kann man das Programm, welches die Daten verwaltet, als Datenbank bezeichnen oder lediglich die Daten selber. Weit verbreitet sind in diesem Zusammenhang die folgenden Bezeichnungen, wie sie zum Beispiel in BRINKHOFF (2005; [6]) dargestellt sind:

Datenbank bezeichnet die Daten selber, die in einer bestimmten Ordnung zum Beispiel auf einer Festplatte abgelegt sind.

Datenbankmanagementsystem ist die Bezeichnung für das Programm, das diese Daten verwaltet, also zuständig ist für die Ablage und die Abfrage der Daten. Anwendungsprogramme greifen nicht direkt auf die Datenbank zu sondern kommunizieren mit dem Datenbankmanagementsystem.

Datenbanksystem ist ein Oberbegriff für die Kombination aus Datenbankmanagementsystem und Datenbank. Es stellt für Anwendungsprogramme ein gekapseltes System dar, in dem Daten gespeichert oder durch Anfragen wieder entnommen werden können.

Der Zusammenhang dieser einzelnen Komponenten ist schematisch in Abbildung 3.1 dargestellt.

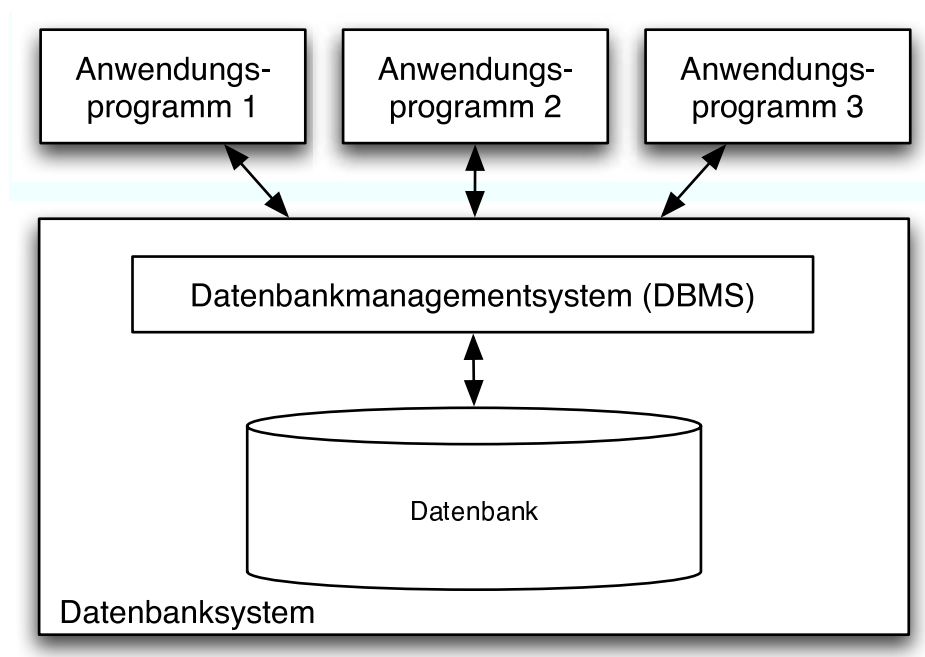


Abbildung 3.1: Aufbau eines Datenbanksystems (nach Brinkhoff, 2005)

3.2 Entwicklung der Datenbanksysteme

Die prinzipiellen Vorteile von heutigen Datenbanksystemen gegenüber einer dateibasierten Datenhaltung wurden schon im einführenden Kapitel beschrieben. Die heute zur Verfügung stehenden leistungsfähigen Datenbanksysteme sind das Ergebnis von jahrzehntelangen Forschungsprojekten. Einen Abriss dieser Entwicklung bieten zum Beispiel LAUSEN UND VOSSER (1996; [28]).

In den späten 60er bis frühen 70er Jahren des vergangenen Jahrhunderts wurden hierarchische- und Netzwerk-Datenbankmodelle entwickelt. Diese werden auch als erste Generation der

Datenbanksysteme bezeichnet und spielen heute keine bedeutende Rolle mehr (SCHEUGENPFLUG, 2005; [51]).

Diese wurden ca. mit Beginn der 80er Jahre von den relationalen Systemen abgelöst. Relationale Datenbanksysteme sind in ihrer Leistungsfähigkeit eine deutliche Erweiterung gegenüber Datenbanksystemen der ersten Generation. Sie sind die heute am weitesten verbreitete Technik und bilden die zweite Generation der Datenbanksysteme.

Parallel zum Einsatz der relationalen Datenbanksysteme wird an der dritten Generation von Datenbanksystemen geforscht, die auch schon produktiv eingesetzt wird. Moderne Programmiersprachen arbeiten objektorientiert und entsprechend wird an Datenbanken zunehmend die Anforderung gestellt, auch Objekte verwalten zu können. Neben echten objektorientierten Datenbanksystemen, die noch nicht weit verbreitet sind, haben sich um Objekt-Datentypen erweiterte relationale Datenbanksysteme bereits etabliert. Diese werden als objekt-relationale beziehungsweise erweiterte relationale Datenbanksysteme bezeichnet.

Da räumliche Daten sich gut als Objekte abbilden lassen, sind Datenbanksysteme der dritten Generation besonders für deren Verwaltung geeignet.

3.2.1 Relationale Datenbanksysteme

Die zweite Generation der Datenbanksysteme – die relationalen Datenbanksysteme – gehen zu einem großen Teil auf die Arbeit von CODD (1970; [11]) zurück, in der die heute noch gültigen Grundprinzipien dieser Technik beschrieben sind.

Datenbanken werden zur Veranschaulichung oft mit Karteikästen verglichen. Die einzelnen Datensätze entsprechen Karteikarten, die wiederum die eigentlichen Informationen (Attribute) beinhalten. Die Analogie eines solchen Karteikastens in relationalen Datenbanken ist eine Tabelle. Der Tabellenkopf enthält die Bezeichnungen der Attribute, die zu einem Datensatz gehören können. Jede Zeile der Tabelle entspricht einem Datensatz und enthält die dazugehörigen Attributwerte.

Das besondere an einer relationalen Datenbank sind die mit namensgebenden Relationen. Dabei handelt es sich um Beziehungen zwischen mehreren Tabellen. Um diese Verknüpfungen zu ermöglichen, ist eine Spalte erforderlich, in der jeder Datensatz eindeutig durch einen Wert repräsentiert ist. Diese Spalte wird als Schlüssel (engl. key) bezeichnet.

Die folgenden Relationen sind zwischen zwei Tabellen A und B möglich:

1:1 Jeweils ein Datensatz der Tabelle A lässt sich eindeutig einem Datensatz der Tabelle B zuordnen.

1:n Ein Datensatz der Tabelle A lässt sich mehreren Datensätzen der Tabelle B zuordnen.

m:n Mehrere Datensätze der Tabelle A lassen sich mehreren Datensätzen der Tabelle B zuordnen.

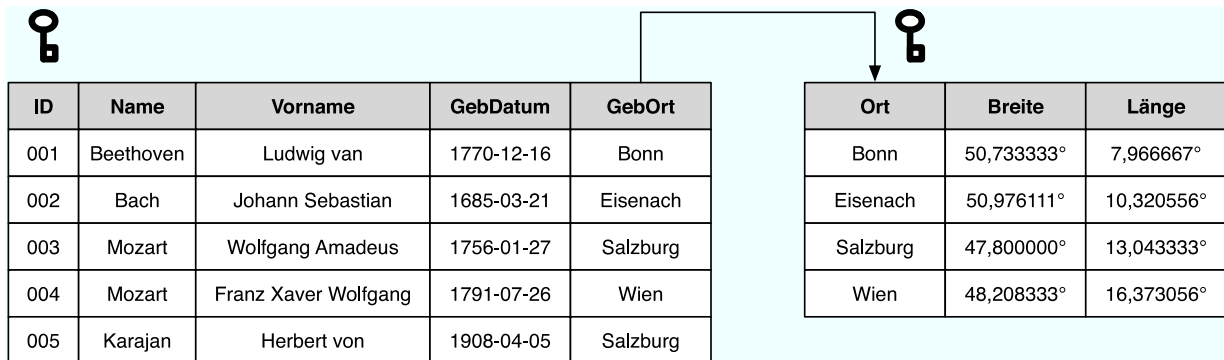


Abbildung 3.2: Beispiel einer Relationalen Datenbank: Die Beispiel Datenbank enthält zwei Tabellen, die über eine 1:n Relation miteinander verknüpft sind. Jede der Tabellen hat einen Primärschlüssel.

In dem Beispiel in Abbildung 3.2.1 sind zwei Tabellen dargestellt. In der linken sind Personen mit ihrem Geburtsdatum und Geburtsort hinterlegt. Die rechte Tabelle enthält Orte mit ihren Koordinaten in Dezimalgrad. Beide Tabellen haben jeweils einen Schlüssel (🔑), über den die Zeilen eindeutig definiert sind. In der ersten Tabelle ist dies eine künstliche laufende Nummer während in der zweiten Tabelle die Städtenamen für eine eindeutige Zuordnung in diesem einfachen Beispiel ausreichen. Über die Spalten **GebOrt** beziehungsweise **Ort** besteht eine Relation zwischen den beiden Tabellen. So kann man zum Beispiel in einer Karte die Geburtsorte der Personen anzeigen. Dargestellt ist eine 1:n Relation. Das bedeutet für diesen Fall, dass mehrere (n) Personen in einem Ort geboren sein können. Andererseits kann eine Person nicht mehrere Geburtsorte haben, was ja auch der allgemeinen Lebenserfahrung entspricht. Die Relation sorgt dafür, dass einer Person aus Tabelle A über den Geburtsort eine Koordinate zugeordnet werden kann.

Neben den Grundlagen des relationalen Datenmodells wurden von CODD unter dem Begriff der Normalisierung auch Mechanismen beschrieben, die zur Unabhängigkeit der einzelnen Tabellen und damit zur Vermeidung von Redundanzen führen.

Die Definition von Tabellen erfolgt über eine Datendefinitionssprache. Hierfür hat sich der Standard Structured Query Language (SQL) durchgesetzt. SQL wurde ursprünglich als

ANSI-Standard spezifiziert. Mittlerweile wurde SQL als ISO-Norm verabschiedet. Die Norm ISO/IEC 9075-1999 ([23]) beschreibt SQL-99 (auch SQL-3). Aktuell ist die Version SQL 2003 ([24]), die jedoch von den Datenbanksystemen noch nicht vollständig implementiert wird.

3.2.2 Objektrelationale Datenbanksysteme

Objektrelationale Datenbanksysteme stellen eine Erweiterung von relationalen Datenbanksystemen um Objekt-Datentypen dar. Damit verbunden sind neben den Datentypen auch Zugriffstechniken (z.B. Indexmechanismen).

Objektorientierte Programmiersprachen wie C++ oder Java besitzen mittlerweile eine sehr große Verbreitung. In den Programmen können Daten unterschiedlichster Art als Objekte mit Eigenschaften behandelt werden. Diese können zwar prinzipiell auch in relationalen Datenbanksystemen abgespeichert werden. Dazu müssen sie aber in einen Datentyp umgewandelt werden, der dort zur Verfügung steht, zum Beispiel in ein Binary Large Object (BLOB). Diesen Umwandlungsvorgang nennt man Serialisierung. Ein solches serialisiertes Objekt kann durch Auslesen aus der Datenbank im Programm wieder in ein echtes Objekt umgewandelt werden. Innerhalb der Datenbank sind besondere Eigenschaften der serialisierten Objekte, die zum Beispiel für komplexe Abfrageszenarien nützlich sein können, jedoch nicht direkt verfügbar, wenn sie nicht durch entsprechende Tabellen nachgebildet werden.

Da relationale Datenbanksysteme eine weite Verbreitung haben und bereits große Datenbestände damit verwaltet werden, war und ist man bestrebt, die bestehenden Systeme um objektorientierte Datentypen und Funktionen zu erweitern. Dies führte zu den objektrelationalen Datenbanken.

Mit einem objektrelationalen Datenbanksystem können wie mit einem relationalen Datenbanksystem einfache Datentypen (Text, Zahl, Datum, ...) verwaltet werden. Darüber hinaus können aber auch selbst generierte komplexere Objekte verwaltet werden. Im Rahmen dieser Arbeit sind besonders geometrische Objekte (Punkte, Linien, Flächen, ...) von Interesse, prinzipiell sind aber vielfältige Einsatzszenarien möglich.

Außerdem können mit einem objektrelationalen Datenbanksystem räumliche Abfragen und Analysen bearbeitet werden. Da die Geometrien als Objekte repräsentiert werden, können auch komplexe Funktionen ausgeführt werden. Zum Beispiel können schon vom Datenbanksystem Koordinatentransformationen oder Funktionen wie Buffer oder das Verschneiden von Geometrien durchgeführt werden.

3.3 Vergleich von (objekt-) relationalen Datenbanksystemen

Der Vergleich der „Standard-Funktionen“ eines (objekt-) relationalen Datenbanksystems ist nicht Thema dieser Arbeit. Dennoch ist auch für den Einsatz eines Datenbanksystems zur Verwaltung von Geodaten wichtig, wie leistungsfähig das zugrundeliegende Datenbanksystem ist. Neben den eigentlichen Geometrien, die besondere Mechanismen erfordern, werden in einer räumlichen Datenbank nämlich auch Sachdaten gespeichert, die wie in regulären Datenbanken verwaltet werden.

Funktionen für die Sicherstellung der Datenkonsistenz, wie zum Beispiel Transaktionen, oder zur Verbesserung der Leistungsfähigkeit, wie dem Mechanismus der Replikation, stellen wichtige Auswahlkriterien dar, wenn ein Datenbanksystem für einen konkreten Einsatzzweck evaluiert wird.

Über die grundlegenden Datenbankfunktionen wurden bereits zahlreiche vergleichende Untersuchungen durchgeführt. Die Tabelle 3.3 stellt beispielhaft und knapp zusammengefasst die wichtigsten Parameter für die am weitesten verbreiteten (objekt-) relationalen Datenbanksysteme dar (SNYDER, 2005; [54]). Dabei fällt auf, dass selbst in diesem sehr allgemein gehaltenen Vergleich schon „GIS Capabilities“ als eine Rubrik aufgeführt werden, was deren zunehmende Bedeutung unterstreicht. Allerdings ist der Begriff „GIS Capabilities“ unscharf und ein wirklicher Vergleich dieser Fähigkeiten findet nicht statt.

Dieser allgemeine Vergleich zeigt, dass sich auf dem Papier die freien Datenbanksysteme (Firebird, Ingres, MySQL, PostgreSQL) bezüglich des Funktionsumfangs nicht hinter ihren kommerziellen Gegenstücken (DB2, Oracle, SQL Server) verstecken brauchen. Werden große Datenbankprojekte geplant muss aber dennoch genau geprüft werden, ob die zu erwartenden Anforderungen von einem bestimmten System tatsächlich erfüllt werden können.

Von den kommerziellen Datenbanksystemen gibt es mittlerweile auch teilweise im Leistungsumfang oder in der Lizenzierung abgespeckte Versionen, die ebenfalls kostenlos zur Verfügung gestellt werden.

Der Vergleich der in der Tabelle dargestellten und zum Teil unternehmenskritischen Datenbankfähigkeiten ist nicht das Ziel dieser Arbeit. Auch eine Bewertung der Leistungsfähigkeit (Benchmark) kann an dieser Stelle nicht erfolgen, da die Leistungsfähigkeit stark vom jeweiligen Anwendungsszenarium und der verwendeten Hardware abhängt.

Tabelle 3.1: Vergleich von kommerziellen und Open Source Datenbanksystemen als klassische Datenbank Snyder (2005); DB2 8.2, Firebird 1.5, Ingres r3, MySQL 5.0, Oracle 10g, PostgreSQL 8, SQLServer 2005

	DB2	Firebird	Ingres	MySQL	Oracle	PostgreSQL	SQL Server
Operating System	AIX, HP-UX, Linux, Solaris, Windows	AIX, FreeBSD, HP-UX, Linux, Mac OS X, Solaris, Windows	AIX, HP-UX, Linux, Solaris, Windows	AIX, Linux, FreeBSD, HP-UX, Mac OS X, NetBSD, QNX, Solaris, SCO, OS/2, Novell Netware, Irix, Dec OSF	AIX, HP-UX, IBM z/OS, Linux, Mac OS X, OpenVMS, Solaris, Windows	AIX, BSD/OS, Linux, HP-UX, IRIX, Mac OS X, NetBSD, OpenBSD, Solaris, Tru64 UNIX, UnixWare, Windows	Windows
Transactions	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Replication	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Foreign Keys	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Integrity Constraints	Yes	Yes	Yes	No*	Yes	Yes	Yes
Views	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Sequences	Yes	Yes	Yes	No**	Yes	Yes	Yes
Triggers	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Full Outer Joins	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Row Level Locking	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Multi-Version Concurrency	No	Yes	Yes	Yes	Yes	Yes	Yes
Cursors	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Stored Procedures	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Procedural Language	Visual Basic, .NET, C, C++, C#, COBOL, Java, or REXX	C, C++, Java, PL/SQL-compatible	PL/SQL-compatible	PL/SQL-compatible	PL/SQL	PL/pgSQL, PL/Tcl, PL/Perl, PL/Python, or user defined	Transact-SQL
GIS Capabilities	Yes	No	No	Yes	Yes	Yes	No
Authentication Schemes	password, encrypted password, client, Kerberos, GSS-API	password, encrypted password, ip address	ip address, password, Kerberos	ip address, password, ssl	base, digest, ip address domain-based, password, crypt	Ident, Kerberos, MDS, PAM, password, SSL, trust	mixed-mode, Windows NT mode

NOTES:
 * MySQL solely provides the NOT NULL constraint
 ** MySQL provides the auto_increment column type which is similar but not equivalent to sequences

Vielmehr wird der in diesem Vergleich mit „GIS-Capabilities“ zusammengefasste Teil für eine Auswahl dieser Datenbanksysteme weiter untersucht. Faktoren wie Ausfallsicherheit oder Datenbankgröße werden bewusst außer Acht gelassen.

Vor dem Einsatz eines Datenbanksystems für ein Projekt oder in einem Unternehmen muss eine Evaluation durchgeführt werden, die unter anderem die eigenen Anforderungen sowie die Fähigkeiten von in Frage kommenden Datenbanksystemen umfasst. Für jedes hier aufgeführte Datenbanksystem gibt es Anwendungsbeispiele im unternehmenskritischen Einsatz mit zum Teil sehr großen Datenbanken.

Was die Integration von GIS-Funktionalitäten angeht gelten die freien Datenbanksysteme MySQL und PostgreSQL als am weitesten verbreitet und am weitesten fortgeschritten. Die-

se beiden Datenbanksysteme werden in den folgenden Kapiteln näher vorgestellt. Auffällig bei diesen beiden Datenbanksystemen ist auch die Vielzahl der unterstützten Betriebssysteme, wobei anzumerken ist, dass die Liste der unterstützten Betriebssysteme in Tabelle 3.3 für beide Datenbanksysteme keineswegs vollständig ist. Neben der freien Verfügbarkeit ist diese Plattformunabhängigkeit interessant für viele Einsatzszenarien.

Oracle bietet in seinen Datenbanksystemen die räumlichen Erweiterungen „Oracle Locator“ und darauf aufbauend „Oracle Spatial“ an. Die Erweiterung „Oracle Locator“ soll auch in dem im Leistungsumfang eingeschränkten Produkt Oracle Database XE enthalten sein. Dieses Datenbanksystem wird kostenlos verfügbar sein und auch in kommerziellen Einsatzszenarien verwendet werden dürfen. Durch die geplante Unterstützung von „Oracle Locator“ und der kostenlosen Verwendbarkeit wurde dieses Datenbanksystem in den Vergleich der freien Datenbanksysteme mit aufgenommen, wobei der Begriff „frei“ sich hier auf die Bedeutung „kostenlos“, und nicht auf quelltextoffen bezieht.

Im Gegensatz steht die kostenlose Version der DB2 von IBM (DB2 Universal Database Express Edition) lediglich ohne die räumlichen Erweiterungen zur Verfügung ([21]).

4 Standards als Grundlagen für Geodatenbanksysteme

Damit eine Datenbanksystem Geodaten speichern und über Abfragen wieder zur Verfügung stellen kann müssen zwei wesentliche Bedingungen gegeben sein. Erstens muss die Datenbank technisch in der Lage sein, die Daten zu verwalten und zweitens müssen – möglichst standardisierte – Schnittstellen zur Verfügung stehen, damit mit unterschiedlichen Programmen auf die Daten zugegriffen werden kann.

Für die Standardisierung zum Umgang mit Geodaten ist im wesentlichen das Open Geospatial Consortium (OGC) in Zusammenarbeit mit sonstigen Normungsgremien zuständig. Ein weiteres wichtiges Gremium ist die International Association of Oil & Gas Producers (OGP), die eine Datenbank von Parametern für unterschiedliche geographische Projektionen unterhält.

4.1 Das Open Geospatial Consortium

Das Open Geospatial Consortium (OGC) ist eine gemeinnützige Organisation, die Standards im Bereich räumlicher Daten und ortsbezogener Dienste erarbeitet und betreut. Das OGC arbeitet eng mit internationalen Normungsgremien, vornehmlich der International Organization for Standardization (ISO) zusammen.

Das OGC hat knapp 300 Mitglieder, bei denen es sich zum Großteil um Firmen, Behörden und Universitäten handelt. Das Ziel ist, Standards im GIS-Umfeld zu erarbeiten, die eine Interoperabilität zwischen verschiedenen Produkten ermöglichen ([35]).

Die Interoperabilität von Produkten unterschiedlicher Hersteller sollte zunächst durch gemeinsame Datenformate hergestellt werden, die von möglichst vielen GIS-Programmen unterstützt werden sollten. Es zeigte sich jedoch, dass die Datenformate für die beteiligten Firmen ein wichtiges Mittel zur Abgrenzung von Konkurrenzprodukten waren. Ent-

sprechend schwierig gestalteten sich die Bemühungen um gemeinsame Datenformate, die schließlich mehr oder weniger im Sande verliefen.

Stattdessen stellte sich heraus, dass die Interoperabilität auf der Ebene von gemeinsamen Schnittstellen einfacher zu koordinieren war. Das bedeutet, dass die einzelnen Produkte die Daten wie bisher in eigenen Formaten speichern und bearbeiten. Allerdings stellen sie standardisierte Zugriffsmöglichkeiten auf die Daten zur Verfügung, die Schnittstellen.

Dieser Ansatz stellte sich als sehr erfolgreich heraus und mittlerweile liegen mehrere Spezifikationen vor, die die Interoperabilität von Geodaten erleichtern und sich wachsender Bedeutung erfreuen. Zu nennen wären hier zum Beispiel die Spezifikationen für Web-Map-Server (WMS), Web-Feature-Server (WFS) und der Geography-Markup-Language (GML). Eine Liste der Spezifikationen sowie die Spezifikationen selbst sind beim OGC frei verfügbar ([36]).

Für die Fragestellung von Geodatenbanksystemen besonders interessant ist die Spezifikation „OpenGIS Simple Feature Implementation Specification for SQL“, die bis vor kurzem in der Version von 1999 aktuell war (OPEN GIS CONSORTIUM, 1999; [38]). Diese Version ist die Grundlage für die momentan verfügbaren Implementierungen.

Die Spezifikation beschreibt Schnittstellen, die das Speichern und Abfragen sowie einfache Operationen von Simple Features (Punkte, Linien, Polygone, ...) im Rahmen von SQL-Anfragen erlauben. Es werden mit Well-Known-Text (WKT) und Well-Known-Binary (WKB) Datenformate spezifiziert, in denen Geodaten in die Datenbank eingepflegt oder abgefragt werden können. Außerdem stehen standardisierte Operatoren für Geometrievergleiche und räumliche Abfragen zur Verfügung.

Die folgende Auflistung zeigt eine Auswahl von Produkten, die auf der Grundlage dieser Spezifikation implementiert wurden:

- Oracle Spatial von Oracle
- ArcSDE von ESRI (keine eigene Datenbank; Middleware)
- MySQL von MySQL AB
- PostGIS, (Refractions Research)

In Kapitel 6.5 wird die Unterstützung dieser Spezifikation von verschiedenen freien Implementierungen verglichen.

Im November 2005 wurde in Zusammenarbeit mit der ISO eine überarbeitete Fassung als ISO 19125 (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2004; [25; 26]) und als Simple Features Access (SFA) des OGC herausgegeben (OPEN GEOSPATIAL CONSORTIUM, 2005; [33; 34]). Diese entspricht inhaltlich weitgehend der alten Spezifikation, ist aber zum Teil deutlicher formuliert und besser dokumentiert.

4.2 OGP/EPSSG

Die International Association of Oil & Gas Producers (OGP) ist ein Zusammenschluss von Öl- und Gas produzierenden Unternehmen. Da diese weltweit operieren und mit Daten in unterschiedlichen geographischen Projektionen arbeiten müssen, pflegen sie eine Datenbank mit geographischen Projektionsvorschriften.

Diese Datenbank der OGP wird als EPSG Geodetic Parameter Dataset bezeichnet. Der Name bezieht sich auf die Vorgängerorganisation, der European Petroleum Survey Group (EPSG), die mittlerweile in die OGP überführt wurde.

In der Datenbank sind tausende verschiedene Projektionsparameter jeweils einer eindeutigen Identifikationsnummer, dem EPSG-Code, zugeordnet. Das ermöglicht es, einen Datensatz, für den dieser EPSG-Code bekannt ist, bei Bedarf in eine andere Projektion umzurechnen. Dadurch können Geodaten, die in unterschiedlichen Projektionen vorliegen, miteinander verwendet und gemeinsam dargestellt werden.

Die im vorigen Kapitel vorgestellten OGC- beziehungsweise ISO-Spezifikationen beschreiben für die Geodatenbanksysteme auch einen Mechanismus zu Verwaltung von Projektionsinformationen, die dort als Spatial Reference Identifier (SRID) bezeichnet werden. Einer eindeutigen ID sollen danach Projektionsparameter zugeordnet werden. Die EPSG-Datenbank stellt also eine Implementierung dieses Mechanismus dar.

Die in Deutschland verbreitete Gauss-Krüger Projektion (hier: Zone 3) wird zum Beispiel durch den EPSG-Code 31467 repräsentiert und in der Datenbank durch die folgende Projektionsvorschrift beschrieben:

```
+proj=tmerc +lat_0=0 +lon_0=9 +k=1.000000 +x_0=3500000 +y_0=0 +ellps=bessel  
+datum=potsdam +units=m +no_defs
```

Es handelt sich hierbei um eine Mercator Projektion eines Bessel-Normalellipsoiden und entsprechenden Lageparametern.

Geodatenbanksysteme können zu jedem Datensatz den dazugehörigen EPSG-Code als SRID abspeichern. Sie können dann Geodaten bei Anfragen in verschiedenen Projektionen zurückliefern und Geodaten mit unterschiedlichen Projektionen miteinander vergleichen. Der Mechanismus ist am Beispiel PostgreSQL/PostGIS in Kapitel 7.2 beschrieben.

Die EPSG-Datenbank hat sich als Quasi-Standard etabliert und wird unter anderem von PostGIS unterstützt. Seit Version 10g Release 2 unterstützt auch Oracle Locator, welches die Grundlage von Oracle Spatial bildet, die Verwendung der EPSG-Codes (IHM UND LOPEZ, 2005; BRINKHOFF, 2005; [22; 6]). Die Projektionstabelle, die in den ESRI-Produkten verwendet wird, beruht ebenfalls auf der EPSG-Datenbank.

5 Indexmechanismen zur Verwaltung von Geodaten

Indizes dienen dazu, abgelegte Informationen mit möglichst geringem Aufwand zu finden. Zum Beispiel kann in einem Fachbuch ein Stichwortverzeichnis helfen, relevante Textstellen zu finden, ohne dass dazu das gesamte Buch gelesen werden muss. Dazu wählt man in dem (alphabetisch) sortierten Stichwortverzeichnis einen Eintrag, der der gesuchten Problematik entspricht und sieht dann die Seitenzahlen, auf denen die gewünschte Information steht. Wenn der gesuchte Begriff nicht im Stichwortverzeichnis steht, muss man trotzdem das gesamte Buch lesen, um die gesuchten Informationen zu finden.

Ähnlich wie ein Stichwortverzeichnis soll auch ein Datenbanksystem auf eine Anfrage dazu relevante Daten möglichst schnell zurückliefern. Das bedeutet, dass die Daten nicht bei jeder Anfrage komplett durchsucht werden müssen (sequentielle Suche), sondern durch geeignete Mechanismen die Suche innerhalb eines Teilbereiches der Gesamtdaten ausreicht.

Dabei darf nicht vergessen werden, dass sowohl das Anlegen als auch das Pflegen eines Index unter Umständen einen großen Aufwand erfordern, was wiederum am Beispiel des Stichwortverzeichnisses einleuchtet.

In diesem Kapitel werden zunächst Mechanismen zur Indizierung von nicht-räumlichen Daten vorgestellt. Der zweite Teil beschreibt dann die besonderen Probleme, die bei der Indizierung von räumlichen Daten auftreten und welche Indizierungsmöglichkeiten hierfür zum Einsatz kommen.

5.1 Klassische Datenbankindizes

In „klassischen“ Datenbanken werden überwiegend Daten gehalten, die sich eindeutig in eine alphabetische oder numerische Reihenfolge bringen lassen (alphanumerische Daten, Nicht-Geometrie Daten). Dabei kann es sich zum Beispiel um Namen, Kontonummern, Zeitangaben oder Artikelbeschreibungen handeln.

Wenn eine solche Sortierbarkeit der Daten in einer Dimension gegeben ist, lässt sich ein Index erstellen, über den eine schnelle Suche möglich ist. Ein solcher Index entspricht in etwa einem Zahlenstrahl, auf dem jeder Wert, der indiziert werden soll, eindeutig eingeordnet werden kann. Bei einer Suche wird dann nur der entsprechende Abschnitt des Zahlenstrahls überprüft. Wird ein passender Wert gefunden, ist ein direkter Zugriff auf den referenzierten Datensatz möglich.

Um die Suche innerhalb eines solchen Index zu verkürzen, wird er üblicherweise als sogenannte Baumstruktur realisiert. Die folgende Abbildung zeigt einen B-Baum, wie er erstmals von BAYER UND MCCREIGHT (1972; [4]) beschrieben wurde.

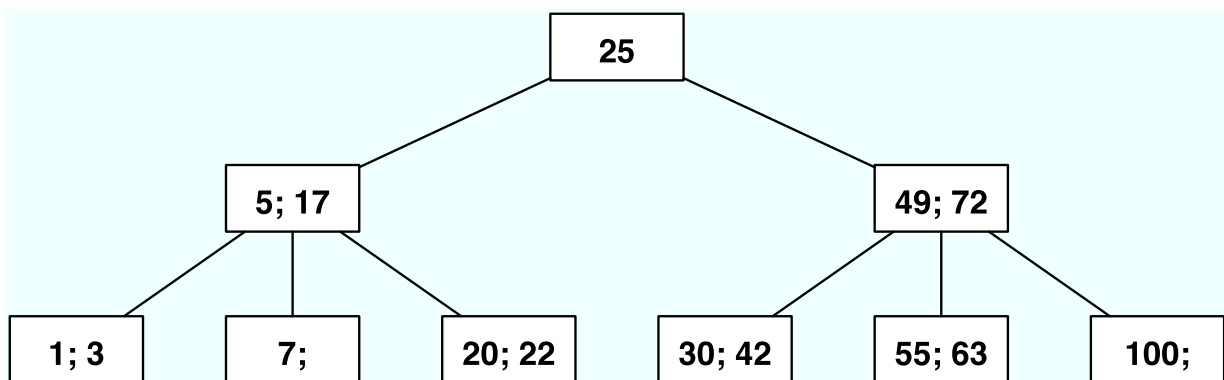


Abbildung 5.1: Darstellung eines B-Baumes, der aus Ganzzahlen aufgebaut wurde. Nähere Erläuterungen im Text.

Ausgehend von der Wurzel (25) sind die Zahlen in diesem Index links eingeordnet, wenn sie kleiner sind als 25 andernfalls rechts. In der nächsten Ebene befinden sich zwei Knoten ([5;17] und [49;72]). Die übrigen Zahlen sind in der nächsten Ebene so sortiert, dass sie kleiner sind als das Intervall (links), innerhalb des Intervalles liegen (mitte) oder größer sind (rechts). Die Knoten dieser letzten Ebene des Baumes werden auch als Blattknoten beziehungsweise Blätter bezeichnet. Die Indexeinträge verweisen auf die entsprechenden Datensätze in der Datenbank.

Werden auf einen solchen Index Suchanfragen ausgeführt, wird ausgehend von der Wurzel ein Suchpfad durchschritten, bis die zur Anfrage passenden Datensätze vorliegen. In dem numerischen Beispiel könnte eine Anfrage zum Beispiel lauten „Liefere alle Werte größer 50“. Da die Zahl 50 größer als der Wert der Wurzel (25) ist, fällt der linke Teil des Baumes vollständig aus der weiteren Suche heraus. Der nächste Knoten des Suchpfades ist deshalb [49;72]. Durch den Vergleich mit 50 fällt hier wiederum der linke Blattknoten weg und als Ergebnis auf die Anfrage verbleiben [55;63;72;100].

Für jede Tabellenspalte, die durchsucht werden soll, muss ein eigener Index angelegt werden. Außerdem ist ein solcher Index nur für Abfragen geeignet, die bereichsweise Vergleiche ermöglichen. Für eine Abfrage nach allen Primzahlen müsste zum Beispiel trotz Index jeder einzelne Datensatz überprüft werden.

Der vorgestellte B-Baum ist eine einfache Variante eines Suchbaumes. Für verschiedene Anwendungszwecke wurden Erweiterungen entwickelt, zum Beispiel der R_+ und der R^* Baum (BECKMANN ET AL., 1990; [5]).

5.2 Räumliche Datenbankindizes

Geodaten haben einige wesentliche Besonderheiten, die sie von alphanumerischen Daten unterscheiden. Die für die Erstellung eines Index wichtigste Eigenschaft ist, dass Geodaten über Geometrien mit Lagebeschreibungen in mindestens zwei Dimensionen verfügen. Das einfache Modell eines eindimensionalen Zahlenstrahls für einen Index funktioniert in zwei Dimensionen nicht mehr, da die Objekte zwei Freiheitsgrade haben und entsprechend nicht nur in einer sondern in zwei Dimensionen einsortiert werden müssen.

Um auch solche Daten indizierbar zu machen und somit Suchzeiten zu erreichen, die besser als eine sequentielle Suche sind, wurden verschiedene Ansätze entwickelt. Die wichtigsten werden im Folgenden kurz vorgestellt.

5.2.1 Der Quadtree

Ein Quadtree eignet sich für eine Indizierung in zwei Dimensionen und bedient sich einer Baumstruktur mit vier Verzweigungen pro Ebene. Die Fläche, über die der Index erstellt werden soll, wird zunächst in vier Quadranten zerlegt. Ausgehend von der Wurzel verweist die Baumstruktur auf diese vier Quadranten (BRINKHOFF, 2005 [6]).

Die Blätter des Quadtree, die letztendlich die Verknüpfung zwischen der Indexstruktur und der Tabelle herstellen, können nur eindeutige Werte aufnehmen. Entsprechend werden die vier Quadranten so lange rekursiv wiederum in jeweils vier gleichgroße Quadrate zerlegt, bis der Quadrant einen eindeutigen Wert repräsentiert. Das heißt er repräsentiert nur noch einen Punkt, eine Linie oder eine Fläche. Sehr anschaulich können Quadrees zur Indizierung von Rasterinformationen verwendet werden. Die rekursive Unterteilung erfolgt dann so lange, bis einzelne Quadranten Homogenbereiche repräsentieren. Die kleinstmögliche Unterteilung ist in diesem Fall ein Bildpunkt (Pixel).

Der Aufbau eines Quadtree für ein Raster ist in Abbildung 5.2.1 dargestellt. Das vorgegebene schwarz-weiß Raster wird in einem ersten Schritt in vier gleichgroße Quadrate zerlegt. Diese werden im Beispiel in der Reihenfolge unten links, unten rechts, oben links und oben rechts durch die Äste des Baumes abgebildet. Der Bereich links oben ist bereits homogen und wird als Blatt mit der Farbinformation „schwarz“ abgespeichert (dritter Ast). Die übrigen Quadranten sind inhomogen. Folglich werden sie durch Knoten repräsentiert (rote Punkte), die dann weiter unterteilt werden. Die Unterteilung erfolgt so lange, bis die komplette Fläche durch schwarze oder weiße Blätter abgedeckt wird.

Auffällig bei einem Quadtree ist, dass die Blätter wie beim B-Baum nicht auf einem Niveau liegen. Ein Quadtree lässt sich auch für eine dreidimensionale Indizierung erweitern und wird dann als Octtree bezeichnet, da das Gesamtvolumen in acht Würfel unterteilt wird.

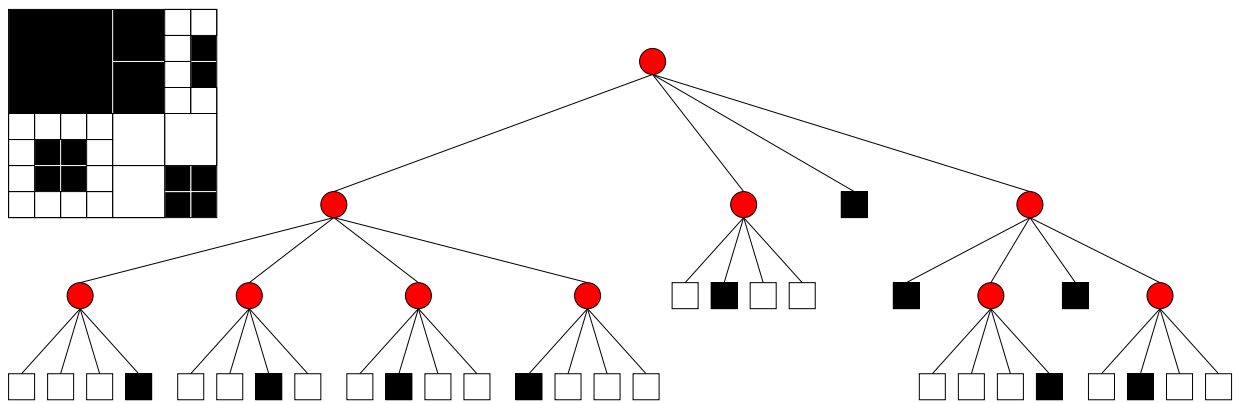


Abbildung 5.2: Darstellung eines Quadtrees: rote Kreise = Knoten, Quadrate = Information (Farbe) der repräsentierten Fläche; Die Beispielaufteilung der Fläche in Quadranten ist unten links, unten rechts, oben links, oben rechts

5.2.2 R-Baum

Grundlage für die meisten heute verwendeten Indexmechanismen zur Indizierung von räumlichen Daten ist der R-Baum, der von GUTTMAN (1984; [1]) erstmals vorgestellt wurde.

Die zugrundeliegende Idee ist, statt der eigentlichen Geometrien die sogenannte Bounding-Box der Geometrien für die Erstellung des Index zu verwenden. Die Bounding-Box ist das kleinste Rechteck, das die Geometrie umgibt. Die Bounding-Boxen lassen sich wesentlich einfacher systematisch ablegen und schnell durchsuchen, da sie durch zwei diagonal gegenüberliegende Eckpunkte eindeutig definiert sind.

Dafür nimmt man in Kauf, dass bei Abfragen über einen solchen Index eventuell zu viele Objekte zurückgeliefert werden, da die Bounding-Box größer als das eigentliche Objekt ist (konservative Approximation; Brinkhoff, 2005 [6]). Deswegen wird eine räumliche Suche meistens in zwei Schritten durchgeführt. Die Indexsuche liefert auf eine Suchanfrage eine Auswahl von Features und nur die Geometrien dieser ausgewählten Features müssen dann noch genau verglichen werden (siehe Kapitel 8.3).

Die Wurzel eines R-Baumes umfasst üblicherweise eine Bounding-Box, die dem Extent des Themas, also der maximalen räumlichen Ausdehnung, entspricht. Außerdem sind in der Wurzel Verweise auf die Kindknoten der nächsten Ebene enthalten.

Die Kindknoten stehen jeweils für einen rechteckigen Ausschnitt des darüberliegenden Knotens beziehungsweise der Wurzel und enthalten ihrerseits eine Bounding-Box, die alle eigenen Kindobjekte umschließt sowie Verweise auf diese Kindknoten oder Blattknoten.

Die Blattknoten bilden die letzte Ebene des Suchbaumes. Sie enthalten die Bounding Boxen der eigentlichen Features sowie einen Verweis in die Datenbank, in der die dazugehörigen Geometrie- und Sachdaten gespeichert sind. In einem R-Baum liegen alle Blattknoten in der gleichen Ebene.

In GUTTMAN (1984; [1]) werden sechs Bedingungen definiert, die ein R-Baum erfüllen muss. Sie enthalten auch die in der vorangegangenen Erklärung nicht berücksichtigten Spezialfälle (Wurzel ist ein Blatt etc.):

1. Jeder Blattknoten (leaf node) enthält zwischen m und M Indexeinträge, außer er ist die Wurzel. M ist die maximale Anzahl, die ein Knoten aufnehmen kann und m ist die Mindestanzahl, die ein Knoten enthalten muss.
2. Jeder Indexeintrag in einem Blattknoten besteht aus der Bounding-Box (I) sowie einem Verweis auf die Daten in der Datenbank
3. Alle übrigen Knoten haben zwischen m und M Kinder (children), außer sie sind die Wurzel.
4. Diese Knoten enthalten eine Bounding-Box (I), die alle Bounding Boxen der abhängigen Kindknoten (child nodes) umfasst sowie Verweise auf die Kindknoten
5. Die Wurzel hat mindestens zwei Kinder (children), außer sie ist ein Blatt (leaf).
6. Alle Blätter liegen in der gleichen Ebene (appear on the same level).

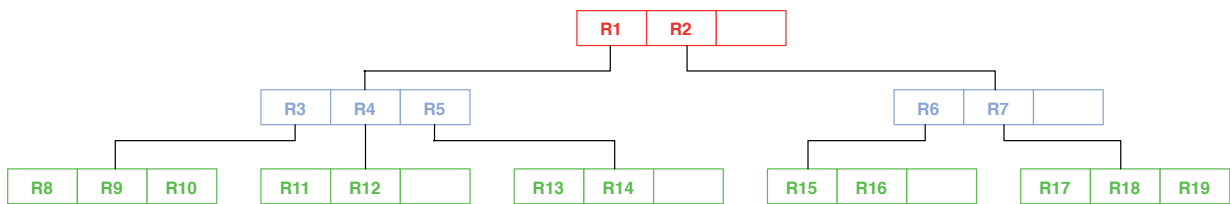


Abbildung 5.3: Darstellung eines R-Baumes. Jede Ebene wird durch eine Farbe dargestellt. Die Wurzel ist Rot und die Blätter sind grün (Abbildung verändert nach Guttman, 1984).

Mit den Parametern m und M kann vorgegeben werden, wie „breit“ und wie „tief“ der Suchbaum sein soll, was wiederum Auswirkungen auf den Overhead der Datenstruktur und den Suchaufwand hat.

Um einen R-Baum zu verwalten werden Funktionen benötigt, um Daten hinzuzufügen, zu entfernen sowie die Baumstruktur zu optimieren. Die erforderlichen grundlegenden Algorithmen sind ebenfalls in GUTTMAN (1984; [1]) dargestellt.

Eine Besonderheit eines R-Baumes ist, dass sich Bounding-Boxen überlappen können. Deswegen ist nicht sichergestellt, dass nur ein Suchpfad ausreicht, um eine Anfrage zu beantworten. Vielmehr kann es erforderlich werden, ab einem bestimmten Knoten oder sogar ab der Wurzel mehrere Pfade zu verfolgen.

Die Abbildung 5.2.2 zeigt den Aufbau eines R-Baumes. In diesem Beispiel beträgt $M=3$ und $m=2$: Jeder Knoten kann maximal drei und minimal zwei Verweise auf Kindobjekte aufnehmen. In keinem Knoten, auch nicht in den Blattknoten, ist die eigentliche Geometrie hinterlegt. Vielmehr repräsentiert ein Knoten jeweils nur die Bounding-Boxen der auf ihn folgenden Knoten sowie Verweise auf diese. Die Blattknoten enthalten jeweils die Bounding-Box der Geometrie, die sie vertreten sowie einen Verweis auf die eigentliche Geometrie in der Datenbank. In Abbildung 5.2.2 sind die Bounding-Boxen und die zugrundeliegenden Geometrien dargestellt, aus denen der Baum generiert wurde.

Die unterschiedlichen Farben der Bounding-Boxen entsprechen den Ebenen der Knoten, die sie vertreten (Abbildung 5.2.2). Gut zu erkennen ist, dass sich die Bounding-Boxen von Knoten einer Ebene überschneiden können (Abbildung 5.2.2 a). Die Blattknoten (grün) sind die Bounding-Boxen der eigentlichen Geometrien (Abbildung 5.2.2 b).

Wie für den B-Baum gibt es auch für den R-Baum Modifizierungen des ursprünglichen Konzeptes. Hier sind vor allem der R^* (BECKMANN ET AL., 1990; [5]) und der $R+$ (SELLIS ET AL., 1987; [53]) Baum zu nennen. Einen Vergleich von weiteren Optimierungsmöglichkeiten bietet LEUTENEGGER ET AL. (1997; [29]). Einen Überblick zur Verwaltung von Geoda-

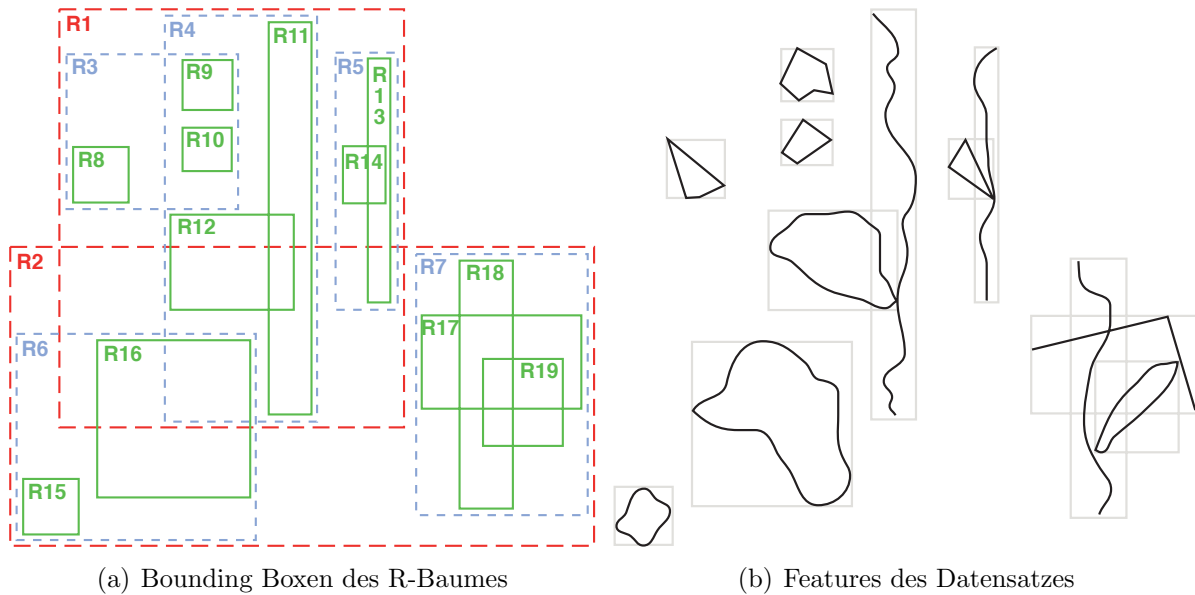


Abbildung 5.4: Darstellung eines R-Baumes. Jede Ebene wird durch eine Farbe dargestellt. Die Wurzel ist Rot und die Blätter sind grün (Abbildung verändert nach Guttmann, 1984).

ten in Geodatenbanksystemen mit einem Ausblick zu Verbesserungsmöglichkeiten bietet GÖBEL (2005; [18]).

Oracle bietet seit Version 8.1.6 für mehrdimensionale Daten eine Implementierung des R-Baumes neben dem bis dahin verwendeten „spatial quadtree index“ (Brinkhoff, 2005; [6]) an. Inzwischen wird die Verwendung des R-Tree empfohlen und ist der Standard-Index-Mechanismus. Der Quadtree soll nur noch in Spezialfällen verwendet werden (MURRAY ET AL. [9]). Auch das freie Datenbanksystem PostgreSQL/PostGIS verwendet für die Verwaltung von Geometrien einen R-Baum (Abschnitt 7.4).

5.2.3 Suche in R-Bäumen

Die räumliche Suche in einem R-Baum nach einem Geometrieobjekt beginnt an der Wurzel. Als Beispiel wird im Folgenden die Suche nach allen Indexeinträgen, also Geometrien, beschrieben, die ganz oder teilweise innerhalb eines Suchfensters liegen.

Ausgehend von der Wurzel wird überprüft, ob die Kindknoten diese Bedingung erfüllen. Die Kindknoten werden durch die Bounding-Box der enthaltenen Geometrien repräsentiert. Es findet also ein Vergleich zwischen dem Suchfenster und diesen Bounding-Boxen statt. Dabei kann das Suchfenster die Bounding-Boxen mehrerer Knoten schneiden, die dann

entsprechend alle durchsucht werden müssen. Außerdem können sich Bounding-Boxen von Knoten einer Ebene, wie bereits beschrieben, ebenfalls überlappen (Abbildung 5.2.3).

Für Kindknoten, die die Suchbedingung erfüllen, wird die Suche mit deren Kindknoten der nächsten Ebene fortgeführt, bis die Ebene der Blattknoten erreicht wird. Kindknoten, die die Suchbedingung nicht erfüllen werden mitsamt den dazugehörigen Kindknoten der nächsten Ebenen von der weiteren Suche ausgeschlossen. Diese Suche muss für alle durch das Suchfenster geschnittenen Knoten durchgeführt werden (mehrere Suchpfade möglich).

Kurz zusammengefasst werden also die Äste von der Wurzel bis zu den Blättern durchsucht solange die Suchbedingung erfüllt ist. In ungünstigen Situationen (Überlappung von Objekten etc.) kann das dazu führen, dass nahezu der gesamte Baum durchsucht werden muss.

Beim Aufbau des Suchbaumes wird deshalb versucht, solche Überlappungen von Objekten weitgehend zu vermeiden. Dies ist jedoch nicht in allen Fällen möglich.

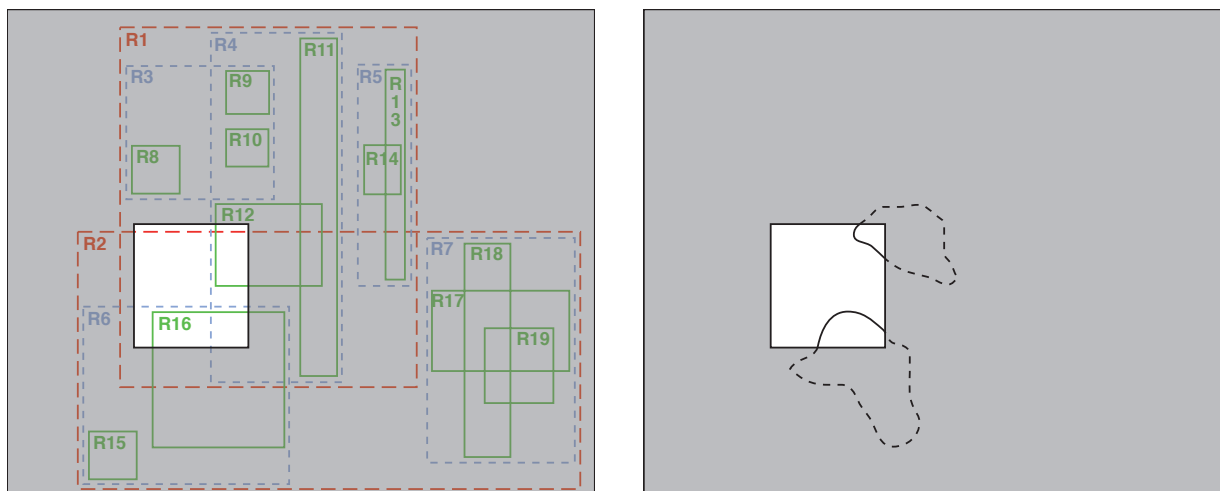
Wenn Objekte zu einer Datenbank hinzugefügt werden, wird jedes Objekt einem Knoten so zugeordnet, dass dessen Bounding-Box nicht oder nur minimal erweitert werden muss. Nach dem Hinzufügen eines Objektes wird dann eventuell eine Reorganisation des Baumes erforderlich, um die vorgegebenen Charakteristika des Baumes (z.B. seine Breite) zu erfüllen.

In Abbildung 5.2.3 ist exemplarisch eine Suche von Geometrien mit Hilfe eines Suchfensters graphisch dargestellt. Der verwendete Baum und die Geometrien entsprechen dem schon in den Abbildungen 5.2.2 und 5.2.2 vorgestellten Beispiel. Grün eingefärbte Knoten müssen in der weiteren Suche berücksichtigt werden während die rot eingefärbten Knoten und ihre Kindknoten sicher ausgeschlossen werden können.

Das weiß hervorgehobene Suchfenster überlappt die Bounding-Boxen der Knoten R1 und R2 (5.2.3 a). Deshalb müssen beide Kindknoten weiter nach passenden Geometrien durchsucht werden.

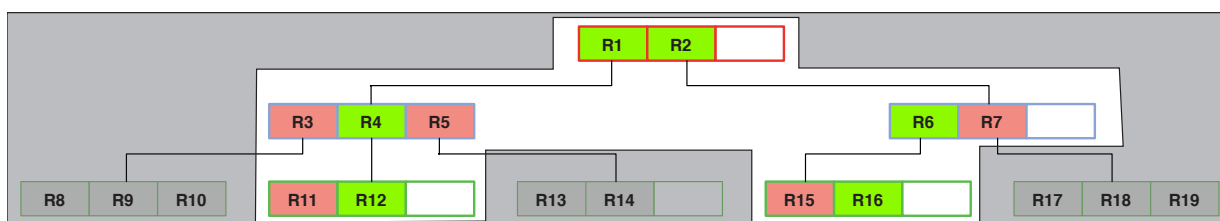
Bei der Überprüfung der nächsten Ebene fallen die Knoten R3, R5 und R7 mit den dazugehörigen Kindknoten aus der Suche heraus, da ihre Bounding-Boxen außerhalb des Suchfensters liegen. Nur die beiden Knoten R4 und R6 dieser Suchebene überlappen noch das Suchfenster.

Die Kindknoten dieser beiden Knoten werden im nächsten Suchschritt wiederum mit dem Suchfenster verglichen. Dabei liegen die Bounding-Boxen der Knoten R11 und R15 außerhalb des Suchfensters, während die Bounding-Boxen von R12 und R16 das Suchfenster



(a) Suchfenster mit Indexstruktur des R-Baumes

(b) Von der Auswahl betroffene Features



(c) Darstellung des angefragten Suchfensters im R-Baum

Abbildung 5.5: Suche in einem R-Baum; Rot hinterlegt: Überprüft aber nicht im Suchfenster, grün hinterlegt: Überprüft und im Suchfenster

überlappen. Diese Knoten bilden die letzte Suchebene (=Blätter) und verweisen auf die eigentlichen Geometrien (5.2.3 b). Um exakte Ergebnisse zu erzielen müssten jetzt noch die so ermittelten Geometrien mit dem Suchfenster verglichen werden, da Objekte außerhalb des Suchfensters liegen können, obwohl ihre Bounding-Box das Suchfenster überlappt. Dieser Fall wird in Kapitel 8.3 im Rahmen der Implementierung einer solchen Suchanfrage erläutert.

In der Baumansicht (5.2.3 c) ist ebenfalls das Suchfenster weiß eingefärbt. Bereiche des Baumes, die für diese Suchanfrage nicht berücksichtigt werden müssen, sind grau ausgeblendet.

Gegenüber einer sequentiellen Suche über alle Geometrien muss in diesem Beispiel also für acht Objekte kein Vergleich durchgeführt werden, da sie durch die Indexstruktur sicher von der Suche ausgeschlossen werden können.

6 Freie Software zur Verwaltung von Geodaten


Wie bereits in Kapitel 3 angedeutet, wurden die beiden Open Source Datenbanksysteme MySQL und PostgreSQL für einen näheren Vergleich ihrer Eignung zur Verwaltung von Geodaten ausgewählt, da sie von Haus aus geometrische Datentypen unterstützen.

Zusätzlich werden die Open Source Projekte JTS und GeoTools vorgestellt, die OGC-Spezifikationen in Java implementieren und als Middleware zum Zugriff auf (fast) beliebige relationale Datenbanksysteme verwendet werden können.

Als Vertreter eines kostenlosen Datenbanksystems, das Geodaten verwalten kann wird Oracle Database XE vorgestellt, das jedoch zum Fertigstellungstermin dieser Arbeit noch nicht in einer finalen Version verfügbar war.

Die Open Source Implementierungen werden in einem abschließenden Abschnitt auf ihre Konformität zur OGC-Spezifikation „Simple Features for SQL“ untersucht und gegenübergestellt.

6.1 MySQL

 MySQL ist das wahrscheinlich bekannteste Open Source Datenbanksystem. Es handelt sich um ein relationales Datenbanksystem, das vor allem im Zusammenspiel mit Skriptsprachen wie PHP und Perl für Internetanwendungen eine große Rolle spielt. MySQL war lange Zeit dafür bekannt, dass zwar nur ein begrenzter Umfang des SQL-Standards unterstützt wurde, dies aber ressourcenschonend und kostenlos. Ende Oktober 2005 wurde MySQL in der Version 5 freigegeben, am weitesten verbreitet ist jedoch noch MySQL 4.x.

Der ursprünglich noch eingeschränkte Funktionsumfang wird nach und nach ausgebaut, um mit anderen relationalen Datenbanksystemen aufzuschließen. In den letzten Versionen wurden so zum Beispiel Stored Procedures, Trigger, Views und Cursor implementiert.

Trotzdem soll MySQL auch weiterhin vor allem ressourcensparend und leicht zu benutzen sein ([32]).

Eine Besonderheit von MySQL ist die Tatsache, dass verschiedene Tabellentypen verwendet werden können. Diese verschiedenen Tabellentypen unterstützen unterschiedliche Fähigkeiten und können deshalb je nach Anforderung gewählt werden. Neben der eigenen MyISAM-Implementierung wurde unter anderem der von InnoDB Oy entwickelte leistungsfähige Tabellentyp InnoDB in MySQL integriert. Die Firma InnoDB Oy wurde zwischenzeitlich von Oracle übernommen. Ob und welche Auswirkungen das auf die weitere Unterstützung von InnoDB durch MySQL hat war zum Zeitpunkt der Erstellung dieser Arbeit noch nicht klar.

Für diese Arbeit ist MySQL interessant, da seit Version 4.1 auch begonnen wurde, Unterstützung für räumliche Daten direkt in das Datenbanksystem einzubauen. Zunächst wurden die erforderlichen Datentypen gemäß der OGC Spezifikation (OPEN GIS CONSORTIUM, 1999; [38]) sowie einfache räumliche Abfragemöglichkeiten zur Verfügung gestellt. Es ist davon auszugehen, dass sich die Unterstützung im Laufe der nächsten Versionen weiter verbessern wird.

Bis zur Version 5.0.16 bestand die Unterstützung räumlicher Daten nur für die eigenen MyISAM-Tabellen. Seitdem wird die Unterstützung auf auch andere Tabellentypen ausgeweitet. Diese Arbeit beschränkt sich auf die Betrachtung des MyISAM Tabellentyps, da hier die Unterstützung am weitesten fortgeschritten ist.

6.1.1 Geschichte

Die Gründer der Firma MySQL AB arbeiteten ursprünglich daran, das seit 1994 bestehende Datenbanksystem Mini SQL (mSQL) zu erweitern. Sie entwickelten dazu den ISAM-Tabellentyp. Aus ihrer Sicht war jedoch mSQL nicht leistungsfähig und erweiterbar genug, um ihren neuen Tabellentyp voll zu unterstützen. Deshalb entwickelten sie zu dem Tabellentyp auch eine SQL-Schnittstelle und hatten schließlich ein vollständig selbst entwickeltes Datenbanksystem, dessen Schnittstellen aber noch weitgehend kompatibel zu mSQL waren.

In der Zwischenzeit wurden weitere Tabellentypen hinzugefügt, die mit MySQL verwaltet werden können. Hierzu gehören zum Beispiel die Typen InnoDB und Berkeley DB (BDB).

6.1.2 Lizenzen

Die Datenbank MySQL wird hauptsächlich von der schwedischen Firma MySQL AB entwickelt und unter zwei unterschiedlichen Lizenzmodellen verbreitet. Je nach Anwendungs-

fall kann beziehungsweise muss der Anwender die für ihn passende Lizenz wählen (MySQL AB, 2004; [31]):

Kommerzielle Lizenz: Für Unternehmen, die die Datenbanktechnologie von MySQL in eigenen Produkten verwenden wollen, aber nicht bereit sind die Quelltextänderungen der Öffentlichkeit zugänglich zu machen, bietet die Firma MySQL AB eine kommerzielle Lizenz an. Ein wesentlicher Punkt ist auch, dass für diesen Lizenztyp von der Firma MySQL vollständiger Support gewährleistet wird.

Open Source Lizenz: MySQL steht außerdem unter der GPL zur Verfügung, mit den in Kapitel 2.4.1 angegebenen Einschränkungen. Darüber hinaus erweitert die Open Source Lizenz von MySQL die Nutzbarkeit auch auf Lizenzen, die ohne diesen Zusatzpassus nicht mit der GPL kompatibel wären (FOSS Exception und Optional GPL License Exception for PHP).

6.1.3 Eignung für Geodaten

MySQL unterstützt die grundlegenden Datenformate, die in der Simple Features Specification erläutert sind. Die Analyseaufgaben sind jedoch noch rudimentär. Sie sind lediglich für die Bounding-Boxen und nicht für die eigentlichen Geometrien implementiert. Auch verschiedene Koordinatensysteme werden nicht unterstützt. Eine einfache R-Baum implementierung steht als Indexmechanismus zur Verfügung.

Genauere Angaben über die Fähigkeiten zur Verwaltung von Geodaten sind in Abschnitt 6.5 dargestellt.

6.2 PostgreSQL / PostGIS



PostgreSQL ist ein objekt-relationales Datenbanksystem. Als weiteres Open Source Datenbanksystem steht es etwas im Schatten des weiter verbreiteten und bekannten Datenbanksystems MySQL. Das hat aber vermutlich auch mit dem (erfolgreichen) Marketing der Firma MySQL AB für MySQL zu tun. Vom Funktionsumfang her braucht sich PostgreSQL auf jeden Fall nicht vor MySQL und auch nicht vor kommerziellen Datenbanken zu verstecken.

PostgreSQL war prinzipiell schon in frühen Versionen in der Lage, Geodaten zu verwalten. Allerdings verfügt das Datenbanksystem ohne die PostGIS Erweiterung nur über rudimentäre Datentypen und Abfragemöglichkeiten für Geodaten.

Über die Erweiterung PostGIS, die im vor allem von der Firma Refractions Research ([48]) entwickelt wird, kann die Datenbank jedoch so erweitert werden, dass die OGC-Spezifikation „Simple Features for SQL“ unterstützt werden. Außerdem wird durch die Erweiterung ein speziell für Geodaten optimierter R-Baum Index zur Verfügung gestellt. Die Einbindung der Erweiterung erfolgt über den Mechanismus prozeduraler Sprachen, über den der Kern der Datenbank erweiterbar ist. So wird die Erweiterung PostGIS über die prozedurale Sprache PL/pgSQL in die Datenbank eingebunden. Wie eine PostgreSQL-Datenbank um die Fähigkeiten von PostGIS erweitert wird wird in Kapitel 7 dargestellt. Dort wird auch näher auf die Indexmechanismen eingegangen, die in PostgreSQL und PostGIS zur Verfügung stehen.

Für die Erstellung dieser Arbeit wurde PostgreSQL in der Version 8.0 verwendet. In der Zwischenzeit wurde die Version 8.1 veröffentlicht, die insbesondere eine Verbesserung des Index Mechanismus enthält (Kapitel 7.4). Diese soll eine deutlich bessere Performance bei Schreibvorgängen bieten. Diese aktuelle Version konnte für die Arbeit nicht mehr berücksichtigt werden.

6.2.1 Geschichte

PostgreSQL hat eine lange Entwicklungsgeschichte, die bis ins Jahr 1986 zurückreicht (POSTGRESQL GLOBAL DEVELOPMENT GROUP; [44]). Die Ursprünge von PostgreSQL wurden an der Universität von Kalifornien, Berkeley gelegt. Zunächst entstand als Nachfolger eines bestehenden Datenbanksystems namens Ingres das Programm Postgres (verkürzt für *post Ingres*). Postgres wurde dann bis zum Jahr 1994 als Forschungsprojekt weiterentwickelt, unter anderem, um objekt-relationale Techniken zu erforschen und erproben.

Im Jahr 1995 wurde die Abfragesprache SQL statt der bisher verwendeten POSTQUEL implementiert. Außerdem wurden große Teile des bestehenden Quelltextes neu geschrieben, was auch zu Geschwindigkeitssteigerungen führte. Aufgrund der großen Änderungen wurde das Projekt umbenannt in Postgres95.

Im darauf folgenden Jahr wurde das Programm erneut umbenannt, und erhielt den heute noch verwendeten Namen PostgreSQL. Grund für den erneuten Namenswechsel war, dass der Quelltext als Open Source freigegeben wurde. Seitdem wurden große strukturelle Änderungen am Quelltext durchgeführt und nach und nach kamen noch fehlende Funktionen

Tabelle 6.1: Methoden und Exportformate von PostGIS, die über die OGC-Spezifikation hinausgehen (Auswahl)

Methode	Erläuterung
Zusätzliche Methoden	
Z()	Z-Koordinate des Punktes
area2d(geometry)	Fläche einer Geometrie, entspricht area(geometry)
distance_sphere(point, point)	Oberflächenentfernung zwischen zwei Punkten (Länge/Breite) auf einer Kugel
distance_spheroid(point, point, spheroid)	Oberflächenentfernung zwischen zwei Punkten (Länge/Breite) auf einem Sphäroiden. Der Sphäroid kann definiert werden. Langsamer aber genauer als distance_sphere(point, point)
length3d(geometry)	3D-Länge eines LineString oder MultiLinestring; Analog length2d() für 2D
length_spheroid(geometry, spheroid)	Länge einer Geometrie auf einem Sphäroid
length3d_spheroid(geometry, spheroid)	Länge einer Geometrie auf einem Sphäroid unter Berücksichtigung der Höhe
distance(geometry, geometry)	Geringster Abstand zwischen zwei Geometrien
max_distance (geometry, geometry)	Größter Abstand zwischen zwei Geometrien
perimeter(geometry)	Umfang eines Polygons oder MultiPolygons
perimeter2d(geometry)	
perimeter3d(geometry)	Umfang eines Polygons oder MultiPolygons in 3D
Zusätzliche Exportformate	
AsEWKT(geometry)	Liefert das Geometrieobjekt als „3D WKT“
AsEWKB(geometry)	Liefert das Geometrieobjekt als „3D WKB“
AsSVG(geometry)	Liefert das Geometrieobjekt als SVG
AsGML(geometry)	Liefert das Geometrieobjekt als GML

hinzu.

Die Erweiterung PostGIS wurde erstmals im Jahr 2001 von der Firma Refrations Research veröffentlicht.

6.2.2 Lizenzen

Die Datenbank PostgreSQL selber unterliegt der BSD-Lizenz. Wie in Kapitel 2.4.2 bereits erläutert steht der Quelltext somit allen Interessierten zur Verfügung und darf verändert werden. Außerdem darf Quelltext für eigene Projekte verwendet werden, ohne diese Änderungen wiederum öffentlich zu machen. Die Erweiterung PostGIS wird dagegen unter der

GPL veröffentlicht. Das hat zur Folge, dass Änderungen im PostGIS-Teil auch wieder im Quelltext veröffentlicht werden müssen.

6.2.3 Eignung für Geodaten

PostGIS 1.0.x unterstützt die Version 1.1 der „Simple Features Specification For SQL“ (OPEN GIS CONSORTIUM, 1999; [38]) des Open Geospatial Consortium (OGC). Zum Zeitpunkt der Erstellung dieser Arbeit war der Konformitätstest von Seiten des OGC noch nicht offiziell bestätigt (OPEN GEOSPATIAL CONSORTIUM; [37]).

Über die Unterstützung der Spezifikation hinaus bietet PostGIS viele weitere Operatoren und Funktionen, die räumliche Daten unterstützen. Erwähnenswert ist zum Beispiel die Unterstützung von drei Dimensionen, die auch angepasste Versionen der Ein- und Ausgabeformate WKB (EWKB) und WKT (EWKT) mit sich bringen. Außerdem werden als zusätzliche Ausgabeformate SVG und GML angeboten.

Einige interessante Funktionen, die über die OGC-Spezifikationen hinausgehen sind in Tabelle 6.1 dargestellt.

6.3 JTS und GeoTools

JTS Die JTS (Java Topology Suite) ist ein weiteres Open Source Projekt und wird im wesentlichen von der Firma Vivid Solutions Inc. ([62]) entwickelt und koordiniert. Die Bibliothek wird unter der LGPL-Lizenz veröffentlicht. Dabei handelt es sich jedoch nicht um ein vollständiges Datenbanksystem mit Unterstützung für räumliche Daten sondern wie der Name schon andeutet um eine in Java programmierte Bibliothekssammlung, die räumliche Datentypen sowie die dazugehörigen Operatoren zur Verfügung stellt und ist damit mehr eine Art Middleware.

Obwohl es sich um kein eigenständiges Datenbanksystem handelt, wird vom Funktionsumfang her die Simple Feature Specification for SQL des OGC erfüllt. Daher kann JTS prinzipiell mit jedem relationalen Datenbanksystem zusammenarbeiten. Für HSQLDB und PostgreSQL gibt es bereits Implementierungen. HSQLDB ist ein in Java implementiertes Datenbanksystem, das zum Beispiel für den Datenbankteil von Openoffice.org verwendet wird.

Die JTS wurde von Java nach C++ portiert und wird unter dem Namen GEOS (Geometry Engine Open Source) als eigenständiges Projekt weiterentwickelt. GEOS wiederum bildet

in PostGIS die Grundlage für die Unterstützung der Simple Feature Specification for SQL. Dieser C++ Port wird im wesentlichen von der Firma Refractions Research betreut, die auch die wesentliche Entwicklungsarbeit der PostGIS-Erweiterung leistet.

GeoTools Die Java Topology Suite bildet unter anderem auch die Grundlage für die GeoTools. Dabei handelt es sich um eine weitere Java-Bibliothekssammlung, die Komponenten zur Verfügung stellt, die die Entwicklung von für spezielle Zwecke optimierten GIS-Applikationen ermöglicht. Das Ziel der GeoTools ist es, die Spezifikationen der OGC zu implementieren und als Komponenten zur Verfügung zu stellen (GEOTOOLS.ORG; [17]).

Unter Anderem ermöglichen die GeoTools den gekapselten Zugriff auf Geodaten aus den verschiedensten Quellen (Dateien, Datenbanken) sowie die Darstellung und Ausgabe von Karten. In Anhang B ist mit Hilfe eines Java-Programmes dargestellt, wie mit Hilfe der GeoTools auf Geodaten, die in einer PostgreSQL/PostGIS Datenbank abgelegt sind, zugegriffen werden kann. Die Abstrahierung der GeoTools ist dabei so groß, dass dieses Programm nur an wenigen Stellen abgeändert werden muss, um auf andere Datenbanksysteme (Oracle, MySQL), Dateien (Shape) oder andere Dienste (WFS) zugreifen zu können.

Die folgenden Vektor-Datenformate werden von den GeoTools unterstützt (GEOTOOLS.ORG; [17]):

- Shapefiles – Dateiformat von ESRI (lesend und schreibend)
- GeoMedia – Dateiformat von Intergraph (lesend)
- GML – Geography Markup Language (lesend)
- WFS – Web Feature Server (lesend und schreibend)
- PostgreSQL/PostGIS (lesend)
- Oracle Spatial (lesend)
- ArcSDE (lesend)
- MySQL (lesend)

6.4 Oracle Database 10g Express Edition

Im Oktober 2005 wurde von Oracle eine Betaversion der Oracle Database 10g als freie Datenbank herausgegeben. Der Begriff frei bezieht sich hierbei auf „kostenlos“. Der Quellcode steht nicht offen zur Verfügung. Die freigegebene Version Oracle Database 10g Express Edition (Oracle Database XE) beruht auf der Codebasis des Produktes „Oracle 10g“ und soll für die Plattformen Linux und Windows erhältlich sein. Es handelt sich nicht um eine Demoversion, die nur getestet werden darf. Vielmehr ist auch eine kommerzielle Nutzung ausdrücklich erlaubt.

Gegenüber von kommerziellen Varianten der Oracle-Datenbanksystemen ist die Version in den folgenden Punkten beschränkt ([43]):

- die Datenbankgröße ist auf 4 GB beschränkt
- es wird auch in Mehrprozessorsystemen nur ein Prozessor verwendet
- die Datenbank verwendet maximal 1 GB Hauptspeicher
- pro Rechner kann nur eine Instanz der Datenbank betrieben werden

Projekte beziehungsweise Datenbanken, die mit diesem Datenbanksystem erstellt werden, können in eine kommerzielle Oracle Datenbank überführt werden, wenn diese Kapazitätsgrenzen nicht mehr ausreichen.

In der Betaversion von Oracle Database XE ist noch keine Unterstützung für räumliche Daten enthalten. Interessant ist jedoch, dass Oracle für die finale Version vorgesehen hat, „Oracle Locator“ zu integrieren. Da diese Datenbank bis zur Fertigstellung dieser Arbeit noch nicht verfügbar war, ist sie in der Vergleichstabelle im nächsten Abschnitt nicht mit aufgeführt. Die Veröffentlichung der endgültigen Version ist für Ende Februar 2006 geplant.

Da die Erweiterung „Oracle Locator“ aus den anderen Oracle-Datenbanksystemen bekannt ist, lässt sich jedoch abschätzen, welche Fähigkeiten zur Verwaltung von Geodaten diese Datenbank in der finalen Version mitbringen wird (Brinkhoff, 2005; [6]). Die erweiterten Fähigkeiten von „Oracle Spatial“ werden weiterhin den kostenpflichtigen Datenbanksystemen von Oracle vorbehalten bleiben.

Mit einer Bewertung dieser Datenbank bezüglich ihrer Eignung zur Verwaltung von Geodaten muss bis zum Erscheinen der finalen Version gewartet werden. Die Restriktionen

bezüglich der verwaltbaren Datenmenge sind so gewählt, dass Projekte auf jeden Fall mit diesem Datenbanksystem evaluiert werden können.

Wächst ein Projekt über diese Restriktionen hinaus geht Oracle davon aus, dass auch die entsprechenden Mittel zur Verfügung stehen, um eine Lizenz für eine kostenpflichtige und nicht beschränkte Version von Oracle zu erwerben. Ein Umstieg auf ein anderes Datenbanksystem ist zu diesem Zeitpunkt dann vermutlich nicht mehr ohne größere Anpassungen durchzuführen, während Oracle einen reibungslosen Übergang von Oracle Database XE auf ein leistungsfähigeres Produkt aus gleichem Hause verspricht.

6.5 Unterstützung der Spezifikation „Simple Features for SQL“ durch Open Source Projekte

Auf der Grundlage der jeweils verfügbaren Dokumentationen für Anwender und Entwickler wurde die folgende Gegenüberstellung der Datenbanksysteme MySQL und PostgreSQL/PostGIS erarbeitet. Dabei stehen nicht die „klassischen“ Anforderungen an eine Datenbank im Mittelpunkt.

Vielmehr spielte für diesen Vergleich die Eignung als Datenbank für räumliche Daten eine Rolle. Als Vergleichskriterium diente die Spezifikation der „Simple Features For SQL“ des OGC (OPEN GIS CONSORTIUM, 1999; [38]). Da diese die Grundlage für die Implementierung der Funktionen war, wurde sie auch für diesen Vergleich herangezogen. Sie wurde zwischenzeitlich durch die neue OGC-Spezifikation (OPEN GEOSPATIAL CONSORTIUM (OGC), 2005; [33; 34]) ersetzt und auch als ISO 19125 veröffentlicht (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO), 2004; [25; 26]). Inhaltlich gibt es für die hier betrachtete Fragestellung jedoch keine Abweichungen.

Die Java Topology Suite (JTS) wurden in den Vergleich mit aufgenommen, da sie die „Simple Features for SQL“ ebenfalls weitestgehend unterstützt und über den Umweg des GEOS-Projektes eng mit der PostGIS Implementierung verwandt ist.

In der Spezifikation beziehungsweise Norm sind zwei große Aufgabenblöcke beschrieben, die eine Datenbank erfüllen muss, damit sie konform ist. So werden zum einen die Geometrietypen und deren Datenformate definiert und zum anderen Funktionen, um räumliche Relationen und Analysen in der Datenbank ausführen zu können.

Der Vergleich von MySQL und PostgreSQL/PostGIS vor dem Hintergrund der Eignung zur Verwaltung von Geodaten zeigt, dass beide die Datenhaltung gemäß der OGC-Spezifikation

Tabelle 6.2: Vergleich der Umsetzung der Simple Features For SQL für ausgewählte Implementierungen (I)

Methode	Erläuterung	MySQL	PostGIS	JTS
Weitere Methoden				
X()	X-Koordinate des Punktes	+	+	+
Y()	Y-Koordinate des Punktes	+	+	+
Length()	Länge des Objekts im dazugehörigen Koordinatensystem	+	+	+
StartPoint()	Liefert den Startpunkt z.B. eines LineString	+	+	+
EndPoint()	Liefert den Endpunkt z.B. eines LineString	+	+	+
IsClosed()	1 (wahr) wenn Startpunkt = Endpunkt	+	+	+
IsRing	1 (wahr) wenn IsClosed() und Geometrie ist „simple“	-	+	+
NumPoints()	Anzahl der Punkte eines Objekts	+	+	+
PointN(n)	Liefert den n-ten Punkt	+	+	+
Area()	Fläche im dazugehörigen Koordinatensystem	+	+	+
Centroid()	Mittelpunkt einer Fläche, kann außerhalb liegen	+	+	+
PointOnSurface()	Punkt der auf der Fläche liegt	+	+	+
ExteriorRing()	„Umrandung“ des Polygons	+	+	+
NumInteriorRing()	Anzahl der eingeschlossenen Ringe	+	+	+
InteriorRingN()	„Umrandung“ des n-ten eingeschlossenen Rings	+	+	+

1: MySQL verwendet GLength(), da Length() bereits verwendet wird

umsetzen. JTS stellt entsprechend der Spezifikation Datenstrukturen zur Verfügung, die das Speichern räumlicher Geometrien erlauben, ist aber für die eigentliche Datenhaltung auf eine darunterliegende relationale Datenbank angewiesen. Eine Bewertung der Performanz der jeweiligen Implementierung ist an dieser Stelle nicht berücksichtigt.

Darüber hinaus unterstützt PostGIS den vollständigen Umfang der räumlichen Relationen und Analysefunktionen, wie sie in der Spezifikation beschrieben sind und geht sogar noch darüber hinaus. So ist zum Beispiel die Arbeit mit dreidimensionalen Koordinaten möglich, was auch in der aktuellen Fassung der OGC-Spezifikation beziehungsweise ISO-Norm noch nicht vorgesehen ist.

Die gängigsten räumlichen Relationen sind in MySQL dagegen nur für den einfachen Vergleich der Bounding-Boxen, aber nicht für die eigentlichen Geometrien implementiert und die räumlichen Analysefunktionen fehlen in der betrachteten und zum Zeitpunkt des Ver-

gleiches aktuellen Version 4.1.x. In der aktuellen Version 5.0 wurden keine weiteren Funktionen zur Unterstützung von Geodaten hinzugefügt.

JTS unterstützt ebenfalls den vollen Umfang der räumlichen Relationen und Analysefunktionen. Einzig die Unterstützung des WKB-Formates fehlt bislang.

Aus dem Vergleich wird deutlich, dass ohne Betrachtung der Leistung sondern rein vom Funktionsumfang her im Bereich der Open Source Software Lösungen zur Verfügung stehen, die die Datenhaltung sowie die Analyse von räumlichen Daten erlauben. Die Kombination PostgreSQL/PostGIS bietet zum gegenwärtigen Zeitpunkt die umfassendste Unterstützung der Simple Features Specification des OGC. MySQL ist für die Datenhaltung räumlicher Daten gerüstet. Weitergehende Funktionen wie räumliche Analysen und Vergleiche müssen aber zum größten Teil außerhalb der Datenbank durchgeführt werden. JTS hat einen anderen Ansatz und stellt zwar räumliche Datentypen zur Verfügung. Diese müssen jedoch anderweitig gespeichert werden. Räumlichen Analysen und Vergleiche werden jedoch vollständig unterstützt. JTS bildet die Basis vieler weiterer Projekte, z.B. Geotools und wird auch in dem GIS-Programm GISterm verwendet, das die Basis für die in die im Rahmen dieser Arbeit programmierte Datenbankanbindung ist.

Von den vorgestellten freien Datenbanksystemen ist die Kombination PostgreSQL/PostGIS am weitesten fortgeschritten, was die Unterstützung räumlicher Daten angeht. So werden nicht nur alle Datentypen und Funktionen zum Umgang mit diesen Daten zur Verfügung gestellt. Auch der optimierte GiST-Index, der von PostGIS beigesteuert wird, ist leistungsfähiger als der einfache von MySQL zur Verfügung gestellte R-Baum.

Aus diesen Gründen wurde die Kombination PostgreSQL/PostGIS ausgewählt, um die Praxistauglichkeit zu erproben.

Tabelle 6.3: Vergleich der Umsetzung der Simple Features For SQL für ausgewählte Implementierungen (II)

Methode	Erläuterung	MySQL	PostGIS	JTS
Geometrie Grundfunktionen				
Dimension()	Dimension des Geometrieobjektes	+	+	+
GeometryType()	Geometrietyp des Geometrieobjektes	+	+	+
SRID()	Code des Koordinatensystems	+ ¹	+ ²	+
Envelope()	Bounding Box des Geometrieobjektes	+	+ ³	+
AsText()	Geometrie als WKT	+	+	+
AsBinary()	Geometrie als WKB	+	+	-
IsEmpty()	1 (wahr) wenn die Geometrie leer ist	+	+	+
IsSimple	1 (wahr) wenn die Geometrie „simple“ ist	○ ⁴	+	+
Boundary()	Liefert die Umgrenzung des Geometrieobjektes	+	+	+
Räumliche Relationen				
Equals(g1, g2)	Zwei Geometrien sind gleich	○ ⁵	+	+
Disjoint(g1, g2)	Zwei Geometrien sind disjunkt	○ ⁵	+	+
Intersects(g1, g2)	Zwei Geometrien überkreuzen sich	○ ⁵	+	+
Touches(g1, g2)	Zwei Geometrien berühren sich	○ ⁵	+	+
Crosses(g1, g2)	Zwei Geometrien überkreuzen sich ohne Berührung	○ ⁵	+	+
Within(g1, g2)	Die erste Geometrie ist in der Zweiten enthalten	○ ⁵	+	+
Contains(g1, g2)	Die zweite Geometrie ist in der Ersten enthalten	○ ⁵	+	+
Overlaps(g1, g2)	Die Geometrien überlappen	○ ⁵	+	+
Relate(g1, g2, p)	p: DE-9IM Matrix wird zum Vergleich herangezogen	-	+	+
Räumliche Analyse				
Distance(g1, g2)	Kürzester Abstand zwischen zwei Geometrien	-	+	+
Buffer(d)	Buffer mit Abstand d	-	+	+
ConvexHull()	Liefert die Konvexe Hülle einer Geometrie	-	+	+
Intersection(g1, g2)	"point set intersection" zweier Geometrien	-	+	+
Union(g1, g2)	"point set union" zweier Geometrien	-	+ ⁶	+
Difference(g1, g2)	"point set differenz" zweier Geometrien	-	+	+
SymDifference(g1, g2)	"point set symmetric difference" zweier Geometrien	-	+	+

1: SRID liefert zwar Werte zurück, MySQL berücksichtigt Projektionen aber nicht

2: verwendet standardmäßig EPSG

3: PostGIS liefert neben xmin/xmax ymin/ymax auch zmin/zmax wenn vorhanden

4: Lediglich Platzhalter, noch nicht implementiert

5: noch nicht implementiert; verweist auf MBR⁵⁰, die die Tests nur für die Bounding Boxen der Geometrieobjekte durchführt (Minimum Bounding Rectangles)

6: bei PostGIS GeomUnion, da UNION ein reserviertes Wort in SQL ist

7 Verwaltung von Geodaten mit PostgreSQL und PostGIS

Als – zumindest gegenwärtig – leistungsfähigstes Open Source Datenbanksystem zur Verwaltung von Geodaten wird in diesem Kapitel am Beispiel PostgreSQL/PostGIS die Erstellung einer Datenbank beschrieben, die räumliche Daten aufnehmen kann.

Außerdem wird gezeigt, wie bestehende räumliche Daten in eine solche Datenbank eingelesen werden können und wie die Daten sowie die Metainformationen (z.B. Geometriotyp, Name, Projektion) in der Datenbank organisiert werden. Auch auf die Generierung neuer Geometrien wird kurz eingegangen.

Damit eine effektive Suche innerhalb der räumlichen Daten möglich wird, wird auch das Anlegen eines räumlichen Index erläutert. Abschließend werden die Indexmechanismen, die PostgreSQL und PostGIS für verschiedene Datentypen zur Verfügung stellen, kurz charakterisiert.

7.1 Anlegen einer Datenbank

Nach der Installation von PostgreSQL muss zunächst ein Benutzer erstellt werden. Danach kann die Datenbank (hier `datenbank`) angelegt werden, in der die folgenden Arbeiten stattfinden. Dies geschieht durch folgende Befehle, die unter UNIX als Administrator in einer Shell (= Eingabeaufforderung unter Windows) eingegeben werden (EISENTRAUT, 2005; [12]):

```
> creatuser benutzer
> createdb -0 benutzer datenbank
```

Der Schalter `-0` gibt den Benutzer an, der der Eigentümer der neuen Datenbank sein soll, hier `benutzer`. Für die weiteren Arbeiten sind keine Administrationsrechte mehr erforderlich. Sie können als `benutzer` ausgeführt werden.

Von Haus aus unterstützt PostgreSQL keine erweiterten Geometrieinformationen. Deshalb muss für Datenbanken, die räumliche Daten aufnehmen sollen, die PostGIS Erweiterung geladen werden:

```
> createlang plpgsql datenbank
> psql -d datenbank -f <Pfad>/lwpostgis.sql
```

Natürlich muss dazu PostGIS auf dem System installiert sein. Der Pfad zu `lwpostgis.sql` (<Pfad>) ist an die lokale Installation anzupassen.

Zunächst wird über den ersten Befehl die Datenbank um die prozedurale Sprache PL/pgSQL erweitert. Danach erfolgt über die Datei `lwpostgis.sql` die eigentliche Einbindung von PostGIS. Diese beiden Schritte haben nur Auswirkungen auf die hier mit `datenbank` bezeichnete Datenbank. Sollen weitere Datenbanken mit dieser Erweiterung versehen werden, müssen diese Schritte für jede weitere Datenbank ebenfalls ausgeführt werden.

Durch das Einbinden von PostGIS werden der Datenbank die beiden Tabellen `geometry_columns` und `spatial_ref_sys` hinzugefügt. Die Tabelle `geometry_columns` enthält Metainformationen über die räumlichen Spalten der in der Datenbank vorhandenen Tabellen. PostgreSQL mit PostGIS Erweiterung kann on-the-fly Konvertierungen von Geometrien in verschiedene Koordinatensysteme durchführen. Es verwendet dafür PROJ.4 sowie die EPSG Datenbank, in der die verschiedenen Koordinatensysteme hinterlegt sind. PROJ.4 ist eine Bibliothek, die ebenfalls als Open Source veröffentlicht wird.

Damit der Datenbank die verschiedenen Projektionen bekannt sind, muss die EPSG Datenbank in die Tabelle `spatial_ref_sys` importiert werden:

```
> psql -d datenbank -f <Pfad>/spatial_ref_sys.sql
```

Nach diesem Schritt sind die Vorbereitungen der Datenbank abgeschlossen und sie ist bereit, um räumliche Daten aufzunehmen. Üblicherweise werden dazu bestehende Daten importiert.

7.2 Import von Geodaten

Mit dem Hilfsprogramm `shp2pgsql`, das zum Umfang von PostGIS gehört, lassen sich Daten im weit verbreiteten Shape-Format (ENVIRONMENTAL SYSTEMS RESEARCH INSTITUTE; [13]) direkt in die Datenbank importieren:

```
> shp2pgsql -s 31467 ShapeName TabellenName datenbank > tmp.sql  
> psql -d shapetest -f tmp.sql
```

Die Daten werden zunächst in eine SQL-Datei geschrieben (`tmp.sql`). Darin sind die SQL-Befehle zum Anlegen einer neuen Tabelle mit den entsprechenden Spalten enthalten (`CREATE TABLE...`) sowie die Befehle, um die Features in die Datenbank einzufügen (`INSERT INTO...`). Durch den Befehl wird neben den Sachdaten auch eine neue Spalte angelegt, die die Geometrieinformationen aufnimmt. Der Parameter `-s` legt die Projektion fest, in der die Daten vorliegen. Der Beispielwert `31467` steht für Gauss-Krüger 3. Streifen. Die Zahlen sind die EPSG-Nummern der entsprechenden Projektion und werden in die Tabelle `spatial_ref_sys` eingetragen. Wird keine Projektion angegeben, wird der Wert beim Import automatisch auf `-1` gesetzt. Die Spalte, die die Geometriedaten aufnimmt heisst standardmäßig `the_geom`.

Die beiden Befehle zum Import von Shapefiles in eine Datenbank lassen sich über eine so genannte Pipe (englisch für Rohr, symbolisiert durch das Zeichen „|“) unter UNIX-Betriebssystemen auch koppeln, so dass der Import ohne eine Zwischendatei ausgeführt werden kann. Mit einer Pipe wird der Ausgabestrom eines Programmes als Eingabe an ein zweites Programm weitergeleitet:

```
> shp2pgsql ShapeName TabellenName | psql -d datenbank
```

Die Tabelle 7.1 zeigt als Beispiel die Datenbanktabelle `geometry_columns` einer PostgreSQL/PostGIS Datenbank nach dem Import verschiedener Datensätze aus Shapedateien mit dem geschilderten Vorgehen. Die Spalte `f_table_name` enthält den Namen des Themas und die Spalte `f_geometry_column` die Spalte, in der die Geometrien der Features abgelegt sind. Die Datenbank unterstützt auch mehrdimensionale Datensätze. Die Anzahl der Dimensionen wird in `coord_dimension` abgespeichert. Der bereits angesprochene EPSG-Code, der die Projektion des Themas widerspiegelt wird in der Spalte `srid` abgespeichert und `type` enthält den Typ (Punkt, Linie, Polygon, ...). Die Typen, die mit einem M enden (z. B. `MULTILINESTRINGM`) können Daten mit X, Y und Z Koordinaten aufnehmen. Diese Typen sind nicht standardisiert sondern eine eigene Implementierung von PostGIS.

Die so angelegte Tabelle entspricht der „Simple Features for SQL“ Spezifikation.

Um effektiv räumliche Suchanfragen ausführen zu können, wird schließlich noch ein räumlicher Index (`spatial index`) für die Geometriespalte angelegt:

Tabelle 7.1: Beispiel für die `geometry_columns` Tabelle einer PostgreSQL/PostGIS Datenbank mit verschiedenen Themen.

oid	f_table_catalog varchar	f_table_schema varchar	f_table_name varchar	f_geometry_column varchar	coord_dimension int4	srid int4	type varchar
70940	"	public	autobahnen	the_geom	2	31467	MULTILINESTRING
70838	"	public	bundeslaender	the_geom	2	31467	MULTIPOLYGON
70889	"	public	bundeslaender_4str	the_geom	2	31468	MULTIPOLYGON
71144	"	public	fluss10_km	the_geom	3	31467	MULTILINESTRINGM
89694	"	public	kreise	the_geom	2	31467	MULTIPOLYGON
59002	"	public	projekte	the_geom	2	-1	POINT
89840	"	public	states	the_geom	2	4326	MULTIPOLYGON
89933	"	public	world94	the_geom	2	4326	MULTIPOLYGON

```
> psql -d datenbank
datenbank=# CREATE INDEX IndexName
           ON TableName
           USING GIST(GeometryField GIST_GEOMETRY_OPS);
datenbank=# VACUUM ANALYZE TableName;
```

Die prinzipielle Funktionsweise dieses Index wird in Kapitel 5.2.2 erklärt. Seit Version 1.0.2 von PostGIS kann mit dem mitgelieferten Tool `shp2pgsql` mit dem Schalter `-I` schon beim Import von Shapefiles automatisch ein GiST-Index erzeugt werden.

Mit dem Befehl `VACUUM` wird die Datenbank optimiert, indem zum Beispiel aus der Datenbank entfernte Daten auch von der Festplatte gelöscht werden. Die Option `ANALYZE` analysiert die Daten einer Tabelle und aktualisiert die Statistiken, um die Abfragen zu optimieren.

Die Optimierung mittels `VACUUM ANALYZE` sollte im produktiven Betrieb von PostgreSQL regelmäßig, zum Beispiel automatisiert, aufgerufen werden. Insbesondere dann, wenn größere Änderungen am Datenbestand (Objekte wurden hinzugefügt, verändert oder gelöscht) durchgeführt wurden.

7.3 Generieren von Geodaten

Da PostGIS die Simple Features Spezifikation unterstützt, können Geometrien auch über die dort beschriebenen Mechanismen direkt durch SQL-Befehle generiert werden. Am anschaulichsten erfolgt die generierung von Geometrien über das Well Known Text Format (WKT). Die folgenden Beispiele wurden der Anleitung für PostGIS entnommen und an die bisher eingeführten Bezeichnungen angepasst ([46]).

```
INSERT INTO datenbank (  
    the_geom, name  
)  
VALUES (  
    GeomFromText('POINT(-126.4 45.32)', 31467), 'Ein Punkt'  
)
```

Das Datum wird in die Datenbank `datenbank` eingefügt. Dabei nimmt dann die Spalte `the_geom` die Geometrie auf, während hier `name` als Beispiel für Nicht-Geometriedaten die Bezeichnung des Objektes aufnimmt. Die Zahl `31467` ist der SRID, also der Code für die verwendete Projektion. Für die Beschreibung der Geometrie kommen die folgenden Typen gemäß Spezifikation in Frage. Die (dreidimensionalen) Typen, die PostGIS zusätzlich zur Verfügung stellt sind nicht mit aufgeführt:

- POINT(0 0)
- LINESTRING(0 0,1 1,1 2)
- POLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
- MULTIPOINT(0 0,1 2)
- MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
- MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
- GEOMETRYCOLLECTION(POINT(2 3),LINESTRING((2 3,3 4)))

7.4 Indizes in PostgreSQL und PostGIS

PostgreSQL bietet verschiedene Indexmechanismen, die für unterschiedliche Datentypen geeignet sind. Darüber hinaus bringt die PostGIS Erweiterung einen eigenen Indexmechanismus mit. Dieser ist speziell für Geodaten optimiert, baut aber auf einem von PostgreSQL bereitgestellten Mechanismus auf.

7.4.1 Indizes in PostgreSQL

Die Datenbank PostgreSQL bietet verschiedene Indexmechanismen an, die jeweils für unterschiedliche Daten geeignet sind ([45]):

B-Bäume

Für Daten, die in einer Dimension sortierbar sind steht eine B-Baum Implementation zur Verfügung. Hierunter fallen die meisten nicht-räumlichen Daten wie Adressen, etc.

R-Bäume

Die aktuelle Version 8.x von PostgreSQL beinhaltet eine Implementierung eines R-Baumes, die jedoch nicht von PostGIS verwendet wird (siehe Kapitel 7.4.2). Für Version 8.2 ist geplant, den R-Baum von PostgreSQL standardmäßig mit Hilfe von GiST abzubilden (siehe nächster Absatz).

GiST

Die Abkürzung GiST steht für Generalized Search Tree. Wie der Name schon sagt, handelt es sich bei dem GiST-Index um einen verallgemeinerten Suchbaum, der für verschiedene Anwendungen angepasst werden kann. Das besondere ist, dass spezialisierte Bäume wie B- und R- Bäume mit Hilfe von GiST implementiert werden können. Wie bereits erläutert ist der von PostGIS verwendete R-Baum eine Implementierung auf der Grundlage des GiST, der den speziellen Anforderungen an die Suche nach georeferenzierten Daten gerecht wird. Die Grundidee des GiST beruht auf der Arbeit von HELLERSTEIN ET AL. (1995; [(19)]). Der Vorteil ist, dass Verbesserungen am GiST auch den darauf aufbauenden Indexmechanismen zugute kommen.

Hash

PostgreSQL stellt auch einen Hash-Index zur Verfügung. Bei einem Hash handelt es sich um eine einfache Datenstruktur, mit der nur die Gleichheit der Einträge mit einer Suchanfrage überprüfbar ist. Anfragen wie zum Beispiel „alle Werte größer 5“ sind damit also nicht möglich. Die Verwendung des Hash-Index wird von den Entwicklern von PostgreSQL nur für Ausnahmefälle empfohlen, da die Abfragen trotz der beschriebenen Einschränkungen überwiegend langsamer als ein B-Baum sind.

7.4.2 Index in PostGIS

Der Aufsatz PostGIS für die Datenbank PostgreSQL bringt neben der Implementierung der in der OGC-Spezifikation beziehungsweise ISO-Norm angegebenen Funktionen und weiteren PostGIS-spezifischen Erweiterungen auch eine eigene Implementierung eines R-Baumes mit. Dieser R-Baum ist mit Hilfe des GiST von PostgreSQL implementiert und hat einige Besonderheiten, durch die er besser für die Indizierung von räumlichen Daten geeignet ist, als der R-Baum, der schon standardmäßig in PostgreSQL enthalten ist.

Gegenüber der R-Baum Implementierung von PostgreSQL sind GiST-Indizes „null safe“, das heißt sie können auch auf Spalten angewandt werden, die leere Zellen enthalten können. Außerdem wird das Konzept der „lossiness“ unterstützt. Für Objekte, die größer als 8K (Kilobyte) sind kann der standard R-Baum von PostgreSQL keinen Index mehr aufbauen. Da für den Aufbau des Index für GIS-Anwendungen nur die Bounding-Box als räumliche Repräsentation erforderlich ist, verwendet die Implementierung von PostGIS auch nur diese. Die genauen Geometriedaten werden wie in Kapitel 8.3 beschrieben erst in der zweite Stufe einer räumlichen Anfrage benötigt.

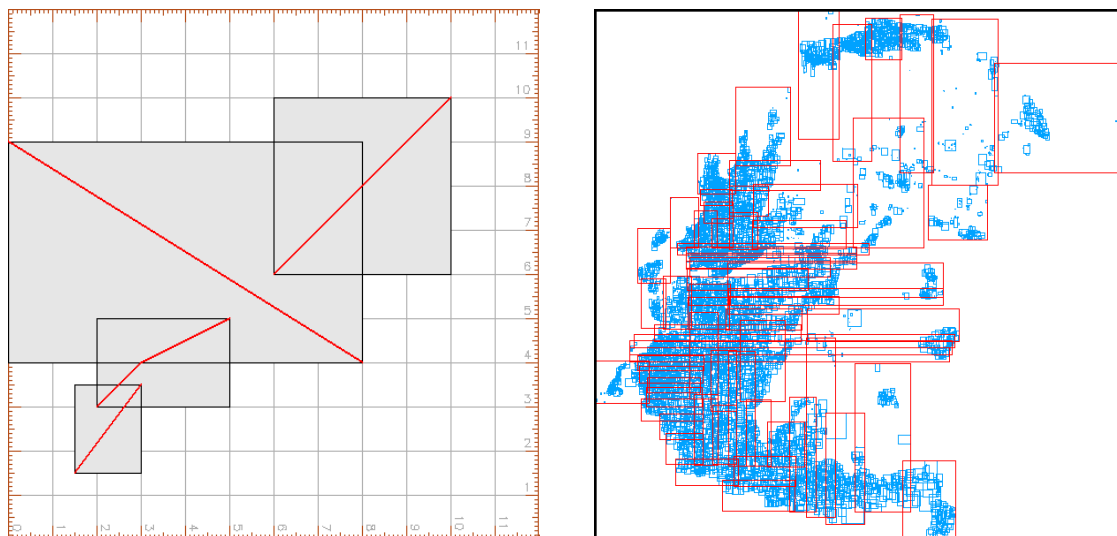


Abbildung 7.1: Beispiele für die Visualisierung eines PostGIS-Index mit der PostgreSQL-Erweiterung „gevel“. Links: Darstellung eines synthetischen Datensatzes (verändert nach HOEFLE [20]), Rechts: Darstellung von Orten in Griechenland (verändert nach BARTUNOV UND SIGAEV [3]).

Dadurch, dass der Index auf dem GiST des darunterliegenden PostgreSQL aufbaut, wirken sich Verbesserungen am GiST-System auch auf die Leistungsfähigkeit von PostGIS aus. So wurde zum Beispiel der GiST-Index für Version 8.1 von PostgreSQL um gleichzeitigen Zu-

griff (concurrency) und Wiederherstellung (recovery) erweitert. Dadurch verbessert sich die Leistung eines Datenbanksystems insbesondere dann, wenn viele Schreib- und Lesezugriffe gleichzeitig bearbeitet werden müssen.

Der Aufbau eines PostGIS Index lässt sich auch visualisieren. Dazu muss die Erweiterung „Gevel“ (BARTUNOV UND SIGAEV [2]) installiert werden. Abbildung 7.1 zeigt links die Bounding-Boxen eines synthetischen Datensatzes mit den dazugehörigen Geometrien. Rechts ist ein Ausschnitt eines R-Baumes mit realen Daten dargestellt. Datengrundlage ist ein Punktdatensatz mit griechischen Orten. In blau und rot sind die Bounding-Boxen von zwei Ebenen des Indexbaumes zu erkennen. In dieser Abbildung wird deutlich, dass sich die Bounding-Boxen von Knoten in einer Ebene (gleiche Farbe) überlappen können, wodurch bei einer Suche im Baum eventuell mehrere Suchpfade durchlaufen werden müssen (Kapitel 5).

8 Datenbankkommunikation über JDBC am Beispiel von PostgreSQL/PostGIS

Die Kommunikation eines Anwendungsprogrammes mit einem Datenbanksystem erfolgt über Schnittstellen. Im Rahmen dieser Arbeit erfolgten Implementierungen von Datenbankverbindungen zu einer PostgreSQL/PostGIS Datenbank in der Programmiersprache Java.

Zunächst wird der Aufbau einer Datenbankverbindung über die JDBC-Schnittstelle beschrieben, über die dann SQL-Anfragen an die Datenbank weitergeleitet werden können.

Aufbauend auf diese Datenbankverbindung wird dann erläutert, wie Metadaten über das Datenbanksystem sowie die enthaltenen Themen abgefragt werden können.

Mit diesen Informationen kann dann die gezielte Abfrage von Geometrieobjekten durchgeführt werden. In diesem Zusammenhang wird auch die Abfrage von Geometrien, die innerhalb eines Suchfensters liegen, unter Berücksichtigung eines räumlichen Index vorgestellt.

In diesem Kapitel werden die einzelnen erforderlichen Schritte abschnittsweise behandelt. Die Abbildung 8.1 stellt schematisch die Komponenten eines Java-Programmes vor, mit dem auf Geometrie- und Sachdaten einer PostgreSQL/PostGIS Datenbank zugegriffen werden kann. Ein Beispielprogramm, das diese Funktionalität zur Verfügung stellt, befindet sich im Anhang A.

8.1 Die Schnittstelle zur Datenbank (JDBC)

Für die Kommunikation mit Datenbanken mit der Programmiersprache Java hat Sun als Entwickler der Sprache die JDBC-Schnittstelle geschaffen (SUN MICROSYSTEMS; [58]). Diese ermöglicht einen weitgehend standardisierten Zugriff auf die zugrundeliegende Datenbank, die somit theoretisch relativ leicht ausgetauscht werden kann. Aktuell ist die Spezifikation für JDBC 3, während die Spezifikation für JDBC 4 bereits vorbereitet wird.

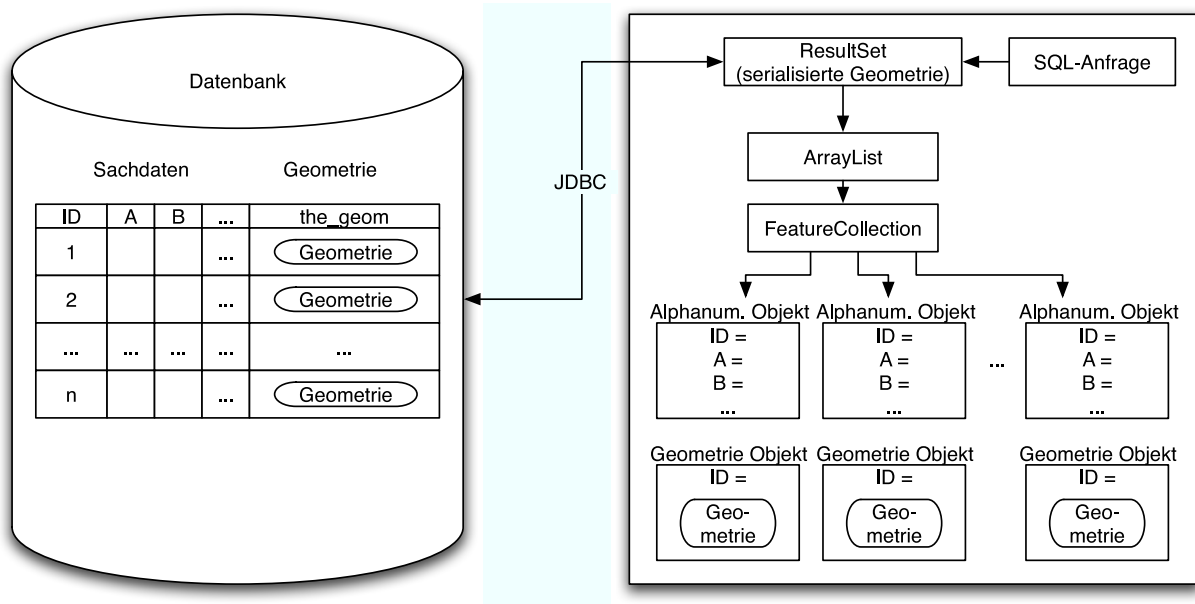


Abbildung 8.1: Schematische Darstellung des Zugriffs auf eine PostGIS Datenbank über die JDBC Schnittstelle.

Für die Implementierung im Rahmen dieser Arbeit wurde der aktuelle JDBC 3 Treiber für PostgreSQL 8 verwendet. Die Abfragen von Metainformationen, Geometrien und Fachdaten erfolgte für dieses Projekt über die JDBC Schnittstelle und entsprechend angepassten SQL-Anfragen.

Um auf die Datenbank zugreifen zu können, muss in dem Java-Programm zunächst der zur Datenbank passende JDBC-Treiber geladen werden. Der Treiber muss sich im Klassenpfad von Java befinden, damit er erkannt wird.

```
Class.forName("org.postgresql.Driver");
```

Wurde der Treiber erfolgreich geladen, kann über die JDBC-Schnittstelle eine Verbindung zur Datenbank aufgebaut werden. Als Parameter müssen der Name der Datenbank sowie gegebenenfalls der Benutzername mit Passwort übergeben werden:

```
String db = "jdbc:postgresql://localhost/datenbank";
String user = "benutzer";
String pw = "";
Connection verbindung = DriverManager.getConnection(db,user,pw);
```

Das Connection-Objekt `verbindung` stellt die Verbindung zur Datenbank dar. Alle weiteren Interaktionen mit der Datenbank werden über dieses Objekt abgewickelt.

8.2 Auslesen von Informationen über die Datenbankverbindung

Allgemeine Informationen über die Datenbank wie ihr Name, der Name und die Version des Treibers etc. können entsprechend dem JDBC-Standard über das Connection Objekt ausgelesen werden:

```
DatabaseMetaData meta = verbindung.getMetaData();
System.out.println(meta.getDatabaseProductName());
System.out.println(meta.getDriverName());
System.out.println(meta.getDriverVersion());
```

Der Befehl `verbindung.getMetaData()` ruft die Metainformationen der Datenbank ab. Als Beispiel werden der Name des Datenbanksystems, der Name des Datenbanktreibers sowie dessen Version ausgegeben.

Für die weitere Arbeit ist diese Art von Meta-Daten aber von untergeordneter Bedeutung. Vielmehr interessiert, welche räumlichen Daten stellt die Datenbank zur Verfügung, handelt es sich um Punkt, Linien oder Flächenthemen, in welcher Projektion liegen sie vor etc.

Wie bereits in Kapitel besprochen verwaltet PostgreSQL/PostGIS diese Informationen in der Tabelle `geometry_columns`. Über eine entsprechende SQL Anfrage lassen sie sich auslesen und auswerten. Im folgenden Beispiel wird in dem String `sqlMetaData` eine SQL-Anfrage formuliert, die für die Tabelle `bundeslaender` die Metadaten aus `geometry_columns` abrufen:

```
Statement stm = verbindung.createStatement();
// Tabellename, für den die Meta-Daten ausgelesen werden:
String tabelle = "bundeslaender"
String sqlMetaData = "SELECT * FROM geometry_columns
                     WHERE f_table_name = '" + tabelle + "'";
ResultSet ergebnis = stm.executeQuery(sqlMetaData);
```

Das Ergebnis der SQL-Anfrage wird in dem `ResultSet ergebnis` abgelegt. Darin sind dann die entsprechenden Informationen (siehe Abbildung 7.1) enthalten und stehen für die weitere Verwendung zur Verfügung.

8.3 Räumliche Ausdehnung und Auslesen der Geometrien innerhalb einer Bounding-Box

Bei einem räumlichen Datensatz interessiert man sich unter anderem auch dafür, wie groß seine räumliche Ausdehnung ist. Üblicherweise wird diese Ausdehnung als das kleinste Rechteck, das alle Features umschließt, definiert und als Extent bezeichnet.

Die entsprechende Anfrage nach dem Extent eines Themas sieht wie folgt aus:

```
String sqlExtent = "SELECT xmin(extent(the_geom)),  
                    xmax(extent(the_geom)),  
                    ymin(extent(the_geom)),  
                    ymax(extent(the_geom))  
                    FROM '" + tabelle + "'";  
ergebnis = stm.executeQuery(sqlExtent);
```

Es wird wieder eine SQL-Anfrage formuliert und man erhält ein `ergebnis`-Objekt (Result-Set) mit der Antwort der Datenbank. Dabei entspricht `the_geom` der Geometriespalte aus der Tabelle `geometry_columns`. Über das `ergebnis` Objekt können dann die minimalen und maximalen XY-Koordinaten des umgebenden Rechteckes ausgelesen werden.

Üblicherweise will man von einer Datenbank bei einer SQL-Anfrage nicht alle enthaltenen Datensätze zurückerhalten. Vielmehr können über den WHERE-Zweig der SQL-Anfrage Bedingungen gestellt werden, welche die Datensätze erfüllen müssen.

Meistens interessiert man sich meistens nur für einen begrenzten räumlichen Ausschnitt. In GIS-Programmen entspricht dieser Ausschnitt dem Kartenteil, der auf dem Bildschirm gezeichnet beziehungsweise gedruckt wird. Dementsprechend ist dieser Ausschnitt – das Suchfenster – rechteckig und kann über die X/Y Koordinaten von zwei diagonal gegenüberliegenden Eckpunkten beschrieben werden.

Die Abbildung 8.3 zeigt Geometrien einer Datenbank, die verschiedene räumliche Lagen zum Suchfenster (weiß) aufweisen. Dargestellt sind Linien (A) und Polygone (B). Für die Geometrien gelten relativ zum Suchfenster die folgenden Beziehungen:

1. Die Geometrie und seine Bounding-Box liegen außerhalb des Suchfensters (1A / 1B)
2. Die Geometrie und somit auch seine Bounding-Box liegen teilweise innerhalb des Suchfensters (2A / 2B)

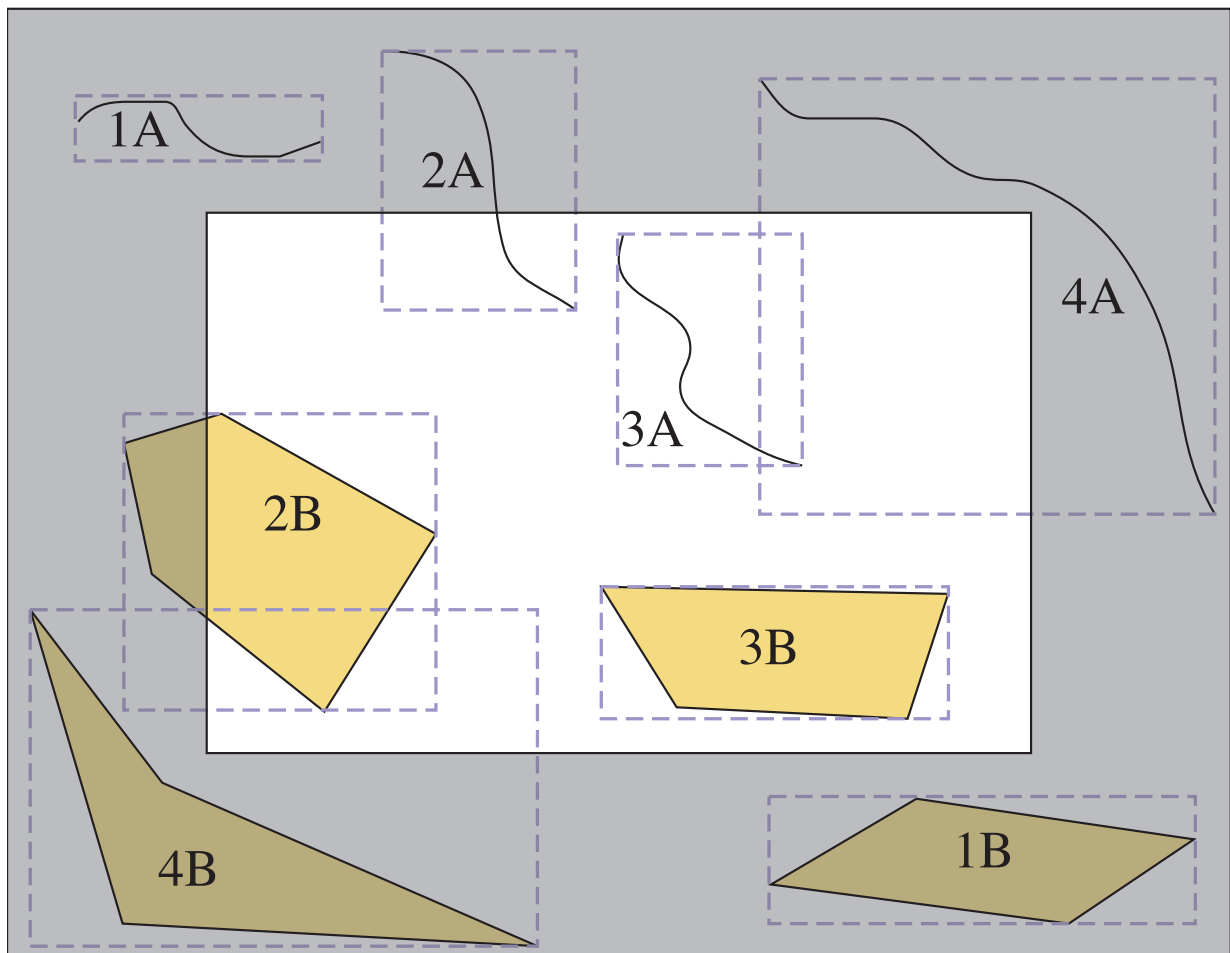


Abbildung 8.2: Suchen von Features innerhalb eines vorgegebenen Extents: Dargestellt sind Features mit ihren Bounding-Boxen. Weiß hervorgehoben ist der Extent, grau sind die Bereiche außerhalb des Such-Extents. Weitere Erläuterungen zur Lage der einzelnen Geometrien im Text.

3. Die Geometrie und seine Bounding-Box liegen komplett innerhalb des Suchfensters (3A / 3B)
4. Die Geometrie liegt außerhalb des Suchfensters die Bounding-Box liegt teilweise darin (4A / 4B)

Anhand des Beispiels wird im Folgenden die zweistufige Abfrage erläutert, die die Objekte zurückliefert, die zumindest teilweise innerhalb des Suchfensters liegen (Intersection).

Das Suchfenster wird wie eine Bounding-Box über zwei diagonal gegenüberliegende Punkte eines Rechteckes definiert. Damit ist es über vier Zahlenwerte (xMin, yMin, xMax und yMax) eindeutig bestimmt.

Alle Features innerhalb des Suchfensters erhält man mit der folgenden Anfrage:

```
String box3d = "BOX3D(" + xmin + " " + ymin + "," +
               + xmax + " " + ymax + ")'::box3d," + srid;
String sql = "SELECT the_geom FROM '" + tabelle + "'
             WHERE the_geom && setsrid('" + box3d + ")
             AND Intersects(the_geom, setsrid('" + box3d + "))";
```

Die Variable `box3d` enthält das Rechteck der Suchanfrage sowie das Koordinatensystem (`srid`), für das die Suchanfrage stattfinden soll. Die Abfrage verläuft in zwei Schritten, um Nutzen aus einem eventuell für die Tabelle vorhandenen räumlichen Index zu ziehen.

Zuerst wird mit

```
the_geom && setsrid('" + box3d + ")
```

mittels des `&&` (overlaps) Operators eine Indexsuche des Suchfensters über die in `the_geom` enthaltenen Geometrien durchgeführt. Da der Index vereinfacht gesagt die Bounding-Boxen der Geometrien enthält, ist dieser Vergleich zwar nicht exakt, weil die Bounding-Boxen (fast) immer größer sind als die Geometrien und liefert deshalb unter Umständen zu viele Ergebnisse. Durch die Verwendung der vereinfachenden Bounding-Boxen statt der wirklichen Geometrien ist der Vergleich aber sehr schnell, da der Index verwendet werden kann.

Mit dem zweiten Teil der Abfrage

```
Intersects(the_geom, setsrid('" + box3d + "))
```

werden dann die tatsächlichen Geometrien der Features aus dieser groben Vorauswahl einzeln überprüft, ob sie innerhalb des Suchfensters liegen. `Intersects` ermittelt, ob sich zwei Geometrien schneiden. In diesem Fall steht `box3d` für das Suchfenster und `the_geom` steht für die in der Datenbank enthaltenen Geometrien.

Diese zweistufige Abfrage ist bei Verwendung eines Index schnell und liefert trotzdem genaue Resultate. Wird diese Genauigkeit nicht benötigt, kann auf den zweiten Teil der Anfrage auch verzichtet werden. Dies kann jedoch dazu führen, dass die Anfrage auch Geometrien zurückliefert, die komplett außerhalb des Suchfensters liegen, weil ihre Bounding-Boxen teilweise das Suchfenster überlappen.

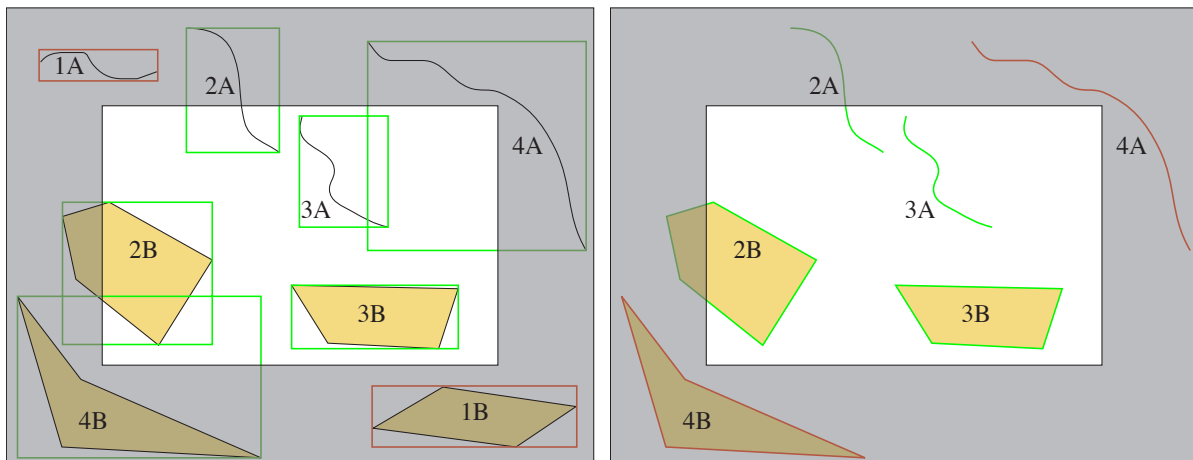


Abbildung 8.3: Die linke Abbildung zeigt die ausgewählten Geometrien nach dem Bounding-Box Vergleich (erster Teil der Abfrage) in grün. Alle Objekte, deren Bounding-Boxen außerhalb des Extents liegen (rot) werden nicht selektiert. Die rechte Abbildung zeigt die ausgewählten Objekte nach dem zweiten Teil der Abfrage. Lediglich Objekte, die tatsächlich wenigstens teilweise innerhalb des Extents liegen werden zurückgeliefert (grün). Objekte, die außerhalb liegen, werden nicht selektiert (rot).

Gemäß OGC-Spezifikation können die Geometriedaten als WKB (Well Known Binary) oder WKT (Well Known Text) abgerufen werden. Standardmäßig gibt PostgreSQL/PostGIS die Daten im WKB-Format zurück. Um das WKT-Format zu erhalten ist in der `SELECT` Anweisung `asText(the_geom)` zu verwenden.

Da PostgreSQL eine objektrelationale Datenbank ist, liegen die Geometrien in der Datenbank als PostGIS-Objekte vor. Diese werden für die Übertragung durch die Umwandlung in WKB oder WKT serialisiert und müssen im Programm durch einen Parser wieder in Geometrieobjekte umgewandelt werden.

8.4 Auslesen der Nicht-Geometrie Daten für die ausgewählten Features

Die Nicht-Geometrie Daten können in einem Schritt zusammen mit den Geometriedaten ausgelesen werden. Dazu sind in der `SELECT` Anweisung einfach wie für eine „normale“ Datenbankabfrage die zusätzlich gewünschten Spaltennamen anzugeben. Das `ResultSet` `ergebnis` enthält dann für alle Features innerhalb des Suchfensters neben der Geometrie auch die dazugehörigen Nicht-Geometrie Daten. Diese können dann durch entsprechende

Objekte abgebildet und zum Beispiel in einer Attributtabelle dargestellt oder für Anfragen verwendet werden.

Es bietet sich an, die Geometrien und die Nicht-Geometrie Daten in getrennten Objekten abzulegen (siehe Abbildung 8) und über eine ID zu Verknüpfen. Zum Zeichnen kann dann auf die reinen Geometriedaten zurückgegriffen. Für die Darstellung von Diagrammen oder Attributtabelle können zusätzlich die Nicht-Geometrie Daten verwendet werden.

9 Implementierung einer PostgreSQL/PostGIS Datenbankanbindung

Bisher wurden mehr oder weniger theoretisch die Fähigkeiten von Open Source Datenbanksystemen vorgestellt, die den Umgang mit räumlichen Daten betreffen.

Als das vielversprechendste Datenbanksystem stellte sich PostgreSQL mit der Erweiterung PostGIS heraus. Nachdem im vorherigen Kapitel einige Grundlagen dieses Datenbanksystems beschrieben wurde – Anlegen einer Datenbank, Einspielen von Daten, . . . – steht in diesem Kapitel die praktische Einsatztauglichkeit im Mittelpunkt.

Dazu wurde eine Anbindung an ein kommerzielles GIS mit den bisher vorgestellten Techniken implementiert. Dieser Prototyp wird im ersten Teil vorgestellt. Der zweite Teil des Kapitels diskutiert Beschränkungen und Verbesserungsmöglichkeiten dieses einfachen Ansatzes.

9.1 Implementierung

Für die Implementierung einer Prototyp-Schnittstelle wurde PostgreSQL in der Version 8.0 sowie PostGIS in der Version 1.0 verwendet. Beide Versionen waren zu Beginn der Arbeit erst seit sehr kurzer Zeit verfügbar und lösen das in vielen Anwendungen bewährte Gespann PostgreSQL 7.4 und PostGIS 0.9 ab. In der Zwischenzeit wurden von beiden Paketen neue Versionen freigegeben (8.1 bzw. 1.1). Diese unterscheiden sich aber nicht in den verwendeten Schnittstellen.

Die Datenbankanbindung wurde für das Programm GISterm der Firma disy Informationssysteme GmbH entwickelt. GISterm ist ein in Java implementiertes GIS, welches unter anderem die bereits vorgestellte Java Topology Suite (JTS) nutzt. Mit dem Programm ist

die Visualisierung, Analyse und Erfassung von Geo- und Sachdaten möglich ([10]). Die Ergebnisse lassen sich anschaulich als Karten und Diagramme darstellen. Das Programm lässt sich eigenständig verwenden oder als Komponente des Berichts- und Auswertesystems *disy Cadenza*.

GISterm ist modular aufgebaut und unterstützt bereits viele verschiedene Datenquellen, wie zum Beispiel Oracle Spatial, ESRI ArcSDE, Shapefiles, WMS, WFS und andere. Zusätzlich zu diesen Datenquellen wurde im Rahmen dieser Arbeit ein Prototyp einer Anbindung von GISterm an eine PostgreSQL/PostGIS Datenbank implementiert. Dabei wurden die in Kapitel 8 vorgestellten Ansätze verwendet. Ein Demoprogramm mit den wesentlichen Fähigkeiten befindet sich im Anhang A.

Da GISterm in Java entwickelt wird, erfolgt die Kommunikation mit der Datenbank über die JDBC-Schnittstelle. Der beschriebene Ansatz, die Kommunikation mit der Datenbank über SQL-Anfragen zu gestalten, kann aber leicht auch mit anderen Programmiersprachen, wie zum Beispiel C++, nachvollzogen werden.

Die Abbildung 9.1 zeigt das Ergebnis der Prototyp-Anbindung. Die dargestellten Geometrien sowie die alphanumerischen Daten stammen aus einer PostgreSQL Datenbank mit PostGIS Erweiterung und können in GISterm dargestellt und ausgewertet werden.

In einem Auswahldialog (links oben) werden die in der Datenbank vorhandenen Themen angezeigt. Das gewünschte Thema kann hier ausgewählt und dann als Thema in GISterm dargestellt werden. Dabei wird unter anderem auch der Typ des Themas (Punkt, Linie, Polygon, ...) übergeben. GISterm generiert daraus einen passenden Legendeneintrag (rechts).

GISterm verwaltet den sichtbaren Kartenausschnitt. Wenn sich dieser durch Verschieben oder Zoomen ändert, wird eine Aktualisierung der sichtbaren Themen angestoßen. Das resultiert in einer Datenbankanfrage nach den Features, die innerhalb des neuen sichtbaren Kartenausschnittes liegen.

Zu den Themen lassen sich auch die Attributtabelle anzeigen, die jeweils die alphanumerischen Sachdaten zu den sichtbaren Objekten enthalten. Die Anzeige der Attributtabelle (unten) wird bei einem Wechsel des Kartenausschnittes dynamisch angepasst. Dargestellt ist in der Abbildung die Attributtabelle für das Thema „kreise“.

Die Sachdaten können in GISterm auch für Selektionen verwendet werden (z.B. Städte > 100.000 Einwohner). Damit diese Selektionen möglich sind, muss dem Datentyp der Datenbank ein Datentyp (String, Integer etc.) innerhalb von GISterm zugeordnet werden. Diese Zuordnung erfolgt ebenfalls durch die Implementierung des Prototypen.

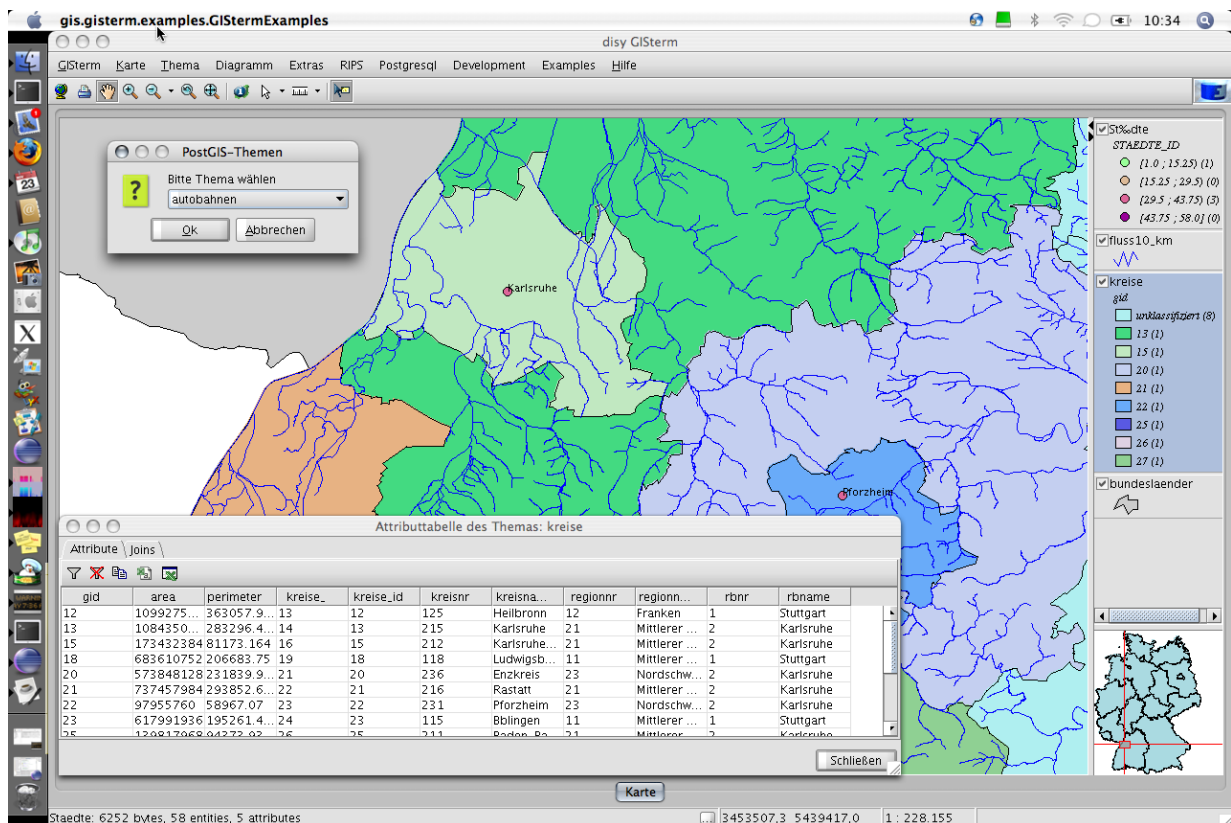


Abbildung 9.1: Prototyp GISterm mit Zugriff auf eine PostgreSQL/PostGIS Datenbank

In einer PostgreSQL/PostGIS Datenbank können auch Themen mit Höheninformationen (z-Koordinate) verwaltet werden. Ein Beispiel für ein solches Thema ist das ebenfalls in der Abbildung dargestellte Flussthema „fluss10_km“. Für die Darstellung in GISterm wird die z-Koordinate ignoriert.

Das Ziel, einen funktionsfähigen Prototypen zu entwickeln, der den Zugriff auf Geodaten in einer PostgreSQL/PostGIS Datenbank erlaubt, wurde mit dem vorgestellten Ansatz erreicht. Alle wesentlichen Anfragemöglichkeiten sind funktionsfähig. Allerdings ist die Implementierung noch nicht so ausgereift, dass sie für einen produktiven Einsatz verwendbar wäre. Einige Probleme sowie Verbesserungsmöglichkeiten werden im nächsten Abschnitt diskutiert.

9.2 Beobachtungen und Weiterführung

Die Implementierung des Prototypen hat gezeigt, dass eine Anbindung einer PostgreSQL/PostGIS Datenbank an ein bestehendes GIS – in diesem Falle GIS der Firma disy Informationssysteme – mit relativ geringem Aufwand durchzuführen ist, wenn das GIS entsprechend modularisiert ist.

Der bisher beschriebene Ansatz ist jedoch für einen produktiven Einsatz noch nicht geeignet. Für einen produktiven Einsatz ist auch eine ausgefeilte Fehlerbehandlung erforderlich. Die momentane Implementierung lässt sich zum Beispiel aus dem Tritt bringen, wenn die zugrundeliegende Datenbank mit bestimmten Codepages angelegt wurde. Die hierfür erforderlichen Fehlerbehandlungen sind zwar für eine tatsächliche Nutzung sehr wichtig. Trotzdem wird in dieser Arbeit auf diese Probleme nicht weiter eingegangen. Vielmehr gibt es im Bereich der verwendeten Technik noch Verbesserungsmöglichkeiten, um die Dateiübertragung und Verarbeitung zu beschleunigen.

Vorteil des gewählten Ansatzes der Datenbankkommunikation über SQL-Anfragen und das WKT-Format als Rückgabeformat ist, dass es sich einfach auch für andere Systeme und Programmiersprachen anpassen lässt.

Bei verschiedenen Tests der Implementierung stellte sich heraus, dass erwartungsgemäß die Verwendung des räumlichen Index zu einer starken Beschleunigung der Anfrage führt. Allerdings wird dieser Geschwindigkeitsvorteil teilweise wieder aufgebraucht, wenn die in Kapitel 8.3 vorgestellte zweistufige Abfrage eingesetzt wird, um nur genau die Features zu erhalten, die tatsächlich im angezeigten Kartenausschnitt liegen. Wird nur der Index verwendet, können zusätzliche Features zurückgeliefert werden, die zwar selber außerhalb des sichtbaren Kartenfensters liegen, deren Bounding-Box jedoch hineinreicht.

Ein Extremfall tritt auf, wenn der gesamte Themenbereich angezeigt wird. Dann werden über die erste Stufe der Anfrage aufgrund der Bounding-Boxen alle Features ausgewählt. Bevor diese Features jedoch von der Datenbank zurückgeliefert werden, wird die wahre Geometrie jedes einzelnen Feature in der Auswahl noch einmal überprüft. In Versuchen mit Datensätzen mit vielen Features ergaben sich große Geschwindigkeitseinbußen, wenn sehr viele oder sogar alle Objekte eines Themas dargestellt wurden.

Eine Möglichkeit zur Beschleunigung könnte also darin bestehen, standardmäßig auf die zweite Abfragestufe zu verzichten und nur den Indexvergleich (Bounding-Boxen) zu verwenden. Für die Darstellung der Geometrien ist das praktikabel. Bei diesem Ansatz werden im Zweifelsfall zu viele Objekte zurückgeliefert, die dann einfach außerhalb des sichtbaren

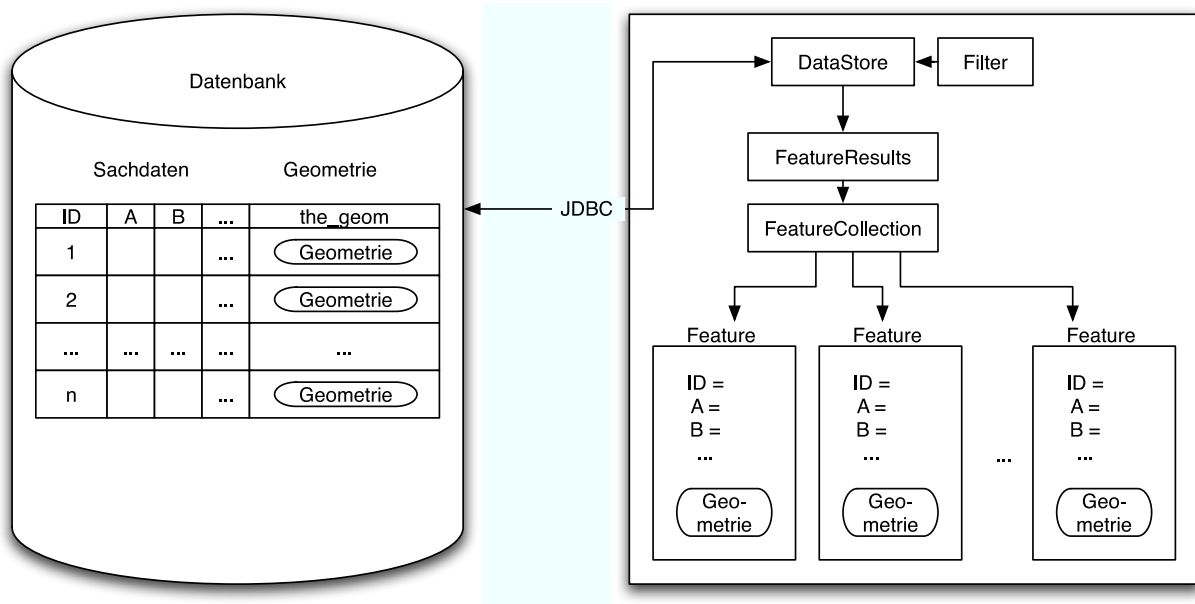


Abbildung 9.2: Schematische Darstellung des Zugriffs auf eine PostGIS Datenbank mit den geotools.

Ausschnittes liegen. Die zweite Abfragestufe muss aber auf jeden Fall zugeschaltet werden, wenn – zum Beispiel für statistische Auswertungen – die exakte Auswahl der Geometrien erforderlich ist.

Außerdem hat sich gezeigt, dass die Verwendung des WKT-Formates bei umfangreichen Themen zu großen Datenmengen führt. Bei einer Anfrage einer Datenbank über ein Netzwerk ist eine starke und lang anhaltende Auslastung des Netzwerkverkehrs zu beobachten. Standardmäßig liefert PostGIS die Daten als WKB zurück, also in einem komprimierten Binärformat. In GIS-Terminologie war jedoch die Übergabe als WKT einfacher durchzuführen, da der PKT-Parser der Java Topology Suite (JTS) verwendet werden kann. Deshalb werden die Daten explizit mit dem Befehl `asWKT()` angefordert.

Ein direkter Zugriff auf die Geometrieobjekte einer Datenbank ist mit der Datenbankreißerweiterung `postgis.jar` möglich, die zu PostGIS gehört. Diese Erweiterung wird auch von den schon vorgestellten geotools genutzt. Durch die Verwendung des geotools-Frameworks lassen sich viele der angesprochenen Probleme vermeiden oder zumindest minimieren. So bieten die geotools eine Kapselung von Datenquellen in Form von DataStores, die bereits über entsprechende Fehlerbehandlungen verfügen. Ein DataStore steht für eine (beliebige) Datenquelle. Über Filter kann definiert werden, welche Daten eine Datenbank zurückliefern soll, die Ergebnisse stehen dann in einer FeatureCollection für die weitere

Verwendung bereit. Ein kurzes Demoprogramm zum Datenbankzugriff über die geotools befindet sich im Anhang B. Die Funktionalität des Programmes entspricht dem Demoprogramm in Anhang A, welches direkt mit SQL-Befehlen arbeitet. Der Zugriff über die geotools ist in Abbildung 9.2 schematisch dargestellt.

Für eine produktive Anbindung sollte die Verwendung der geotools auf jeden Fall in die engere Wahl gezogen werden, da dem Entwickler viele fehleranfällige Entscheidungen abgenommen werden. Nicht zuletzt kann durch das modulare DataStore-Konzept die zugrundeliegende Datenquelle mit relativ geringem Aufwand gewechselt werden. Trotzdem können über Parameter Besonderheiten der zugrundeliegenden Datenquelle genutzt werden. Im Fall von PostgreSQL/PostGIS ist es zum Beispiel möglich, zur Datenübertragung das kompakte WKB-Format zu verwenden.

10 Fazit und Ausblick

Wie der Vergleich des Funktionsumfangs von Open Source Datenbanksystemen mit den entsprechenden Spezifikationen zeigt, gibt es mehrere Open Source Projekte, die sich auf die Verarbeitung von Geodaten verstehen.

Viele davon, insbesondere die vorgestellten Projekte Java Topology Suite (JTS), GEOS, GeoTools und nicht zuletzt PostGIS, stehen dabei in wechselseitigen Beziehungen zueinander und bauen teilweise aufeinander auf.

Das Gespann aus dem objektrelationalen Datenbanksystem PostgreSQL mit der Erweiterung PostGIS gilt als zur Zeit leistungsfähigste Lösung, um mit Open Source Software räumliche Daten zu verwalten. Das weit verbreitete Datenbanksystem MySQL ist von Haus aus ebenfalls in der Lage, räumliche Daten abzulegen. Es bietet aber darüber hinaus nur rudimentäre Abfrage- und Analysemöglichkeiten.

Die Implementierung einer Prototyp-Datenbankanbindung von PostgreSQL/PostGIS an GISern, ein kommerzielles GIS-Programm, zeigt, dass diese Open Source Lösung als Datenquelle für räumliche Daten eingesetzt werden kann. Bis zu einem produktiven Einsatz sind aber noch Optimierungen erforderlich. Vor allem aber muss die Anbindung robuster im Sinne von Fehlertolerant werden. Hierfür bieten die GeoTools, zumindest in der Programmiersprache Java, eine gute Grundlage.

Der Begriff „Freie Datenbanksysteme“ schließt neben den Open Source Lösungen je nach Auslegung auch „kostenlose“ Datenbanksysteme mit ein. Hier plant Oracle mit dem Produkt Oracle XE die Freigabe eines interessanten Vertreters. Oracle XE soll Oracle Locator, und damit eine grundlegende Unterstützung für räumliche Daten, mitbringen. Das mächtigere Oracle Spatial bleibt jedoch weiterhin den kostenpflichtigen Varianten vorbehalten. Zwar ist Oracle XE in der Leistungsfähigkeit künstlich beschnitten. Das System darf aber kommerziell verwendet werden und lässt sich auf kostenpflichtige Versionen migrieren, wenn diese Beschränkungen erreicht werden.

In den nächsten Jahren wird es unabhängig vom Entwicklungs- und Vertriebsmodell zu weiteren technischen Fortschritten im Bereich der Datenbanksysteme kommen, auch was

die Unterstützung von räumlichen Daten angeht. Ob sich objektorientierte Datenbanksysteme durchsetzen werden, steht noch in den Sternen. Auf jeden Fall sind jedoch auch weiterhin Verbesserungen der Indexmechanismen und der Analysemöglichkeiten speziell von räumlichen Daten zu erwarten. Zudem werden mehr und mehr auch Rasterdaten in Datenbanksystemen verwaltet werden. Diese Entwicklung steht noch relativ am Anfang.

Auch im Bereich der Open Source Welt ist in den nächsten Jahren mit einer ständigen Weiterentwicklung und zum Teil auch mit einer Professionalisierung auszugehen. Speziell für den GIS-Bereich seien hier zwei aktuelle Ereignisse beschrieben, die das verdeutlichen.

- Am 4. Februar 2006 wurde die Gründung der Open Source Geospatial Foundation (www.osgis.org) auf einem Treffen in Chicago beschlossen. An dem Treffen nahmen Vertreter von 20 Open Source GIS-Projekten teil. Zu den Gründungsmitgliedern der Organisation werden bekannte Projekte wie GRASS, GeoTools, GDAL, MapServer, ... gehören. Weitere Projekte wollen sich nach der Gründung der Organisation anschließen.

Ziel dieser Organisation ist es, die teils schon vorhandenen Verbindungen zwischen den einzelnen Projekten zu verbessern und weiter auszubauen, so dass eine Suite von Applikationen entsteht, die eine reibungslose Funktionsfähigkeit der einzelnen Komponenten miteinander sicherstellt. Ein weiteres wichtiges Anliegen der Organisation wird es sein, über die gemeinsame Homepage eine zentrale Anlaufstelle zu schaffen. Über ein gemeinsames Marketing verspricht man sich Vorteile in der öffentlichen Wahrnehmung gegenüber unkoordinierten Anstrengungen der Einzelprojekte.

- Mehrere Firmen, die GIS-Lösungen und Dienstleistungen auf der Basis von PostgreSQL/PostGIS anbieten, haben sich im Jahr 2005 zusammengetan, um gezielt die Leistungsfähigkeit in bestimmten Bereichen zu verbessern ([47]). Dazu wurde ein Fonds, in dem letztendlich 8.000 \$ zusammenkamen, eingerichtet. Mit diesem Geld wurden die Entwickler des GiST-Mechanismus dafür bezahlt, dass einige Erweiterungen (recurrency- und recover Unterstützung) bevorzugt implementiert wurden. Dies führte zu einer verbesserten Leistungsfähigkeit des PostGIS-Index, der auf GiST aufbaut.

Die beteiligten Firmen haben dadurch zielgerichtet dafür gesorgt, dass ein von ihnen verwendetes Programm verbessert wurde und die Entwickler haben für diese bevorzugte Implementierung einen Lohn erhalten. Aufgrund der zugrundeliegenden Open Source Lizenz stehen die Änderungen auch der Allgemeinheit zur Verfügung. Dieses Vorgehen, bei dem eine Prämie (engl. bounty) für die Implementierung von

gewünschten Funktionen ausgelobt wird, ist in zunehmendem Maße auch bei anderen Projekten zu beobachten.

Diese Arbeit stellt nur eine Momentaufnahme dar. Schon während des Entstehungsprozesses kamen zum Teil mehrere neue Versionen der unterschiedlichen Programme heraus, die zum Teil deutliche Funktionserweiterungen mit sich brachten. Auch im Bereich der Normungen (z.B. OGC) ist Bewegung. So wurde die ehemalige OGC-Spezifikation „Simple Features for SQL“ in Zusammenarbeit mit der ISO weiterentwickelt. Und schließlich bewegt sich auch die kommerzielle Seite. Die angekündigte, und vorraussichtlich noch im Februar 2006 erscheinende Version der Datenbank Oracle XE zeigt, dass die etablierten Anbieter von kommerziellen Datenbanksystemen durch verstärkte Konkurrenz untereinander und durch Open Source Systeme reagieren müssen. Vor dem Hintergrund ständig wachsender Datenmengen werden die nächsten Jahre zeigen, wie der Markt zwischen den unterschiedlichen Wettbewerbern aufgeteilt wird. Für den Anwender ergibt sich daraus der Vorteil, dass er zwischen verschiedenen leistungsfähigen Alternativen wählen kann.

Abkürzungsverzeichnis

9IM 9 Intersection Method

ANSI American National Standards Institute

BLOB Binary Large Object

BSD Berkeley Software Distribution

C++ Objektorientierte Erweiterung der Programmiersprache C

db2 Name eines Datenbanksystems von IBM

EPSG European Petroleum Survey Group

EULA End User Licence Agreement

EWKB Extended Well Known Binary

EWKT Extended Well Known Text

FOSS Free and Open Source Software

GEOS Geometry Engine Open Source

GIS GeoInformationsSystem

GiST Generalized Search Tree

GML Geography Markup Language

GNU GNU ist Not Unix (rekursive Abkürzung)

GPL General Public License

GRASS Geographic Resources Analysis Support System

ISO International Organization for Standardization

JTS Java Topology Suite

JDBC Java DataBase Connectivity

LGPL GNU Lesser General Public License

MBR minimum bounding rectangle

mSQL Mini SQL

MySQL Freies Datenbanksystem, das auch unter einer kommerziellen Lizenz erhältlich ist

ODBC Open DataBase Connectivity

OGC Open Geospatial Consortium

OPG Association of Oil & Gas Producers

OSI Open Source Initiative

PHP ursprünglich Personal Homepage Tools, mittlerweile PHP Hypertext Preprocessor
(rekursives Akronym)

Pixel Picture Element; Ein Bildpunkt

PostgreSQL Freies Datenbanksystem

PostGIS Erweiterung von PostgreSQL zur Verwaltung räumlicher Daten

PROJ.4 Name eine Open Source Bibliothek zur Konvertierung räumlicher Daten

QGIS Quantum GIS

SQL Structured Query Language

SRID Spatial Reference Identifier

SVG Scalable Vektor Graphics

WKB well known binary

WKT well known text

Anhang A

Das folgende Java-Programm stellt eine Verbindung zu einer PostgreSQL/PostGIS Datenbank über die JDBC-Schnittstelle her. Die Kommunikation erfolgt mit SQL-Anweisungen.

Nach dem Herstellen der Verbindung werden zunächst Informationen über die Datenbank und den Treiber ausgegeben. In einem Auswahlfenster werden dann die in dieser Datenbank vorhandenen Themen angezeigt. Für ein gewähltes Thema werden dann Metainformationen wie zum Beispiel Geometriotyp, Anzahl der Dimensionen und das Koordinatensystem und schließlich der Extent angezeigt. Außerdem werden alle Spaltenbezeichnungen ausgegeben.

Als nächstes werden von der Datenbank alle Features angefordert, die innerhalb einer Bounding-Box liegen. In dieser einfachen Implementierung wird der Extent des Themas als Bounding-Box verwendet. Deshalb liefert die Anfrage alle Features des Themas zurück. Die verwendete SQL-Befehlsfolge wird ebenfalls ausgegeben, gefolgt von der Anzahl der ausgewählten Features.

Die zurückgelieferten Features, könnten dann in dem Programm weiterverarbeitet und zum Beispiel auf dem Bildschirm ausgegeben werden.

Für eine Beispieldatenbank, aus der eine Thema mit den Grenzen der Bundesländer von Deutschland ausgewählt wurde, liefert das Programm die folgende Ausgabe:

```
- Informationen zum Datenbanksystem
  Name: PostgreSQL
  Treibername: PostgreSQL Native Driver
  Treiberdetails: PostgreSQL 8.0 JDBC3 with SSL (build 312)
- Informationen zum gewählten Thema
  Gewähltes Thema: bundeslaender
  Spalte mit Geometriedaten: the_geom
  Geometriotyp: MULTIPOLYGON
  Dimension: 2
  Koordinatensystem (EPSG): 31467
```

- Extent des Themas
 - X min/max: 3280387.0 / 3921542.0
 - Y min/max: 5237512.0 / 6103327.5
- Spaltennamen
 - gid bl count shn gen rau isn the_geom
- SQL Anfrage: SELECT gid, asText(the_geom) AS the_geom_WKT FROM bundeslaender WHERE the_geom && setsrid('BOX3D(3280387.0 5237512.0, 3921542.0 6103327.5) '::box3d,31467) AND Intersects(the_geom, setsrid('BOX3D(3280387.0 5237512.0,3921542.0 6103327.5) '::box3d,31467))
- Anzahl der Features innerhalb der Bounding-Box: 16

```
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JOptionPane;

/**
 * Aufbau einer Datenverbindung über jdbc
 *
 * @author Christian Schanz
 */
public class PostGisRaw {

    public static void main(String[] args) {

        // Verbindungsdaten
        String db = "jdbc:postgresql://localhost/tmp";
        String user = "uaie";
        String pw = "";

        // Treiber laden
        try {
            Class.forName("org.postgresql.Driver");
        } catch (Exception e) {
```


Anhang A

```
System.err.println("Kann_Treiber_nicht_laden");
}

try {
    // Verbindung zur Datenbank herstellen
    Connection verbindung = DriverManager.getConnection(db, user, pw);
    // Informationen zur Datenbank
    DatabaseMetaData meta = verbindung.getMetaData();
    System.out.println("-_Informationen_zum_Datenbanksystem");
    System.out.println("_Name:" + meta.getDatabaseProductName());
    System.out.println("_Treibername:" + meta.getDriverName());
    System.out.println("_Treiberdetails:" + meta.getDriverVersion());

    /*
     ** Tabellen mit Geometriedaten abfragen und Auswahl treffen
     */
    Statement statementThemeSelect = verbindung.createStatement();
    String sqlThemeSelect = "SELECT_*_FROM_geometry_columns_ORDER_BY_
        f_table_name";
    ResultSet resultThemeSelect = statementThemeSelect
        .executeQuery(sqlThemeSelect);

    List geometryThemesTMP = new ArrayList();
    while (!resultThemeSelect.isLast()) {
        resultThemeSelect.next();
        geometryThemesTMP.add(resultThemeSelect
            .getString("f_table_name"));
    }
    String[] geometryThemes = new String[geometryThemesTMP.size()];
    for (int i = 0; i < geometryThemesTMP.size(); i++) {
        geometryThemes[i] = (String) geometryThemesTMP.get(i);
    }

    Object selectedTheme = JOptionPane.showInputDialog(null,
        "Bitte_Thema_wählen", "PostGIS-Themen",
        JOptionPane.QUESTION_MESSAGE, null, geometryThemes,
        geometryThemes[0]);

    /*
     ** Meta-Daten für gewünschten Datensatz aus geometry_columns
     auslesen
     ** (=> Alle Tabellen mit Geometriedaten)
     */
}
```

```

Statement statementMetaData = verbindung.createStatement();
String sqlMetaData = "SELECT * FROM geometry_columns WHERE
    f_table_name = '
    + selectedTheme.toString() + '";
ResultSet resultMetaData = statementMetaData
    .executeQuery(sqlMetaData);

resultMetaData.next(); //Gehe zum ersten (und einzigen) Datensatz
String f_table_name = resultMetaData.getString("f_table_name");
String f_geometry_column = resultMetaData
    .getString("f_geometry_column");
String type = resultMetaData.getString("type");
int coord_dimension = resultMetaData.getInt("coord_dimension");
int srid = resultMetaData.getInt("srid");
System.out.println("- Informationen zum gewählten Thema");
System.out.println("  Gewähltes Thema: " + f_table_name);
System.out.println("  Spalte mit Geometriedaten: "
    + f_geometry_column);
System.out.println("  Geometrietyp: " + type);
System.out.println("  Dimension: " + coord_dimension);
System.out.println("  Koordinatensystem (EPSG): " + srid);

/*
  ** Extent des Themas bestimmen
  */
String sqlExtent = "select xmin(extent(" + f_geometry_column
+ ")), xmax(extent(" + f_geometry_column
+ ")), ymin(extent(" + f_geometry_column
+ ")), ymax(extent(" + f_geometry_column + ")) from "
+ f_table_name + ";";
resultMetaData = statementMetaData.executeQuery(sqlExtent);
resultMetaData.next();
System.out.println("- Extent des Themas");
System.out.println("  X_min/max: "
    + resultMetaData.getDouble("xmin") + " / "
    + resultMetaData.getDouble("xmax"));
System.out.println("  Y_min/max: "
    + resultMetaData.getDouble("ymin") + " / "
    + resultMetaData.getDouble("ymax"));

/*
  ** Meta-Daten der nicht-Geometrie Spalten:
  */

```

```

Statement statementNonGeometry = verbindung.createStatement();
String sqlNonGeometry = "SELECT_*_FROM_" + f_table_name
+ "_LIMIT_0";
ResultSet resultNonGeometry = statementNonGeometry
.executeQuery(sqlNonGeometry);

// Spaltennamen des Themas
System.out.println("-_Spaltennamen");
ResultSetMetaData metaTab = resultNonGeometry.getMetaData();
for (int i = 1; i <= metaTab.getColumnCount(); i++) {
    System.out.print("_" + metaTab洗getColumnName(i));
}
System.out.println();

// Alle Features innerhalb einer Bounding-Box auswählen

Statement statementGetFeatures = verbindung.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);

// Die SQL Anfrage verwendet in einem ersten Schritt (EE) die
// BoundingBox
// für eine Grobauswahl => schnell durch Index
// Genaue Abfrage dann mit Intersects()

// Teilstring BOX3D
String box3d = "BOX3D(" + resultMetaData.getDouble("xmin") + "_"
+ resultMetaData.getDouble("ymin") + ","
+ resultMetaData.getDouble("xmax") + "_"
+ resultMetaData.getDouble("ymax") + ")'::box3d," + srid;

// Teilstring für abzufragende Spalten zusammenbauen
// Die ID-Spalte (gid) wird als erste Spalte eingefügt
// Die Geometrien liegen in f_geometry_column
String columnNames = "gid,_";

columnNames = columnNames + "asText(" + f_geometry_column
+ ")_AS_the_geom_WKT";

// SQL Abfrage zusammenbauen
String sqlGetFeatures = "SELECT_" + columnNames + "_FROM_"
+ f_table_name + "_WHERE_" + f_geometry_column
+ "_&_setsrid(' + box3d + ")_AND_Intersects("

```

Anhang A

```
+ f_geometry_column + ",setsrid('" + box3d + "))";
System.out.println("-SQL-Anfrage:" + sqlGetFeatures);

// SQL Anfrage abschicken
ResultSet resultGetFeatures = statementGetFeatures
    .executeQuery(sqlGetFeatures);
resultGetFeatures.last();
System.out
    .println("-Anzahl der Features innerhalb der Bounding-Box:"
        + resultGetFeatures.getRow());

// In resultGetFeatures liegen die Geometrien im WKT-Format
// zusammen
// mit einer eindeutigen ID und können weiter verwendet werden

} catch (Exception e) {
    System.out.println("Exception" + e);
}

}
}
```

Anhang B

Das folgende Java-Programm stellt eine Verbindung zu einer PostgreSQL/PostGIS Datenbank mit Hilfe der GeoTools her. In einem Auswahlfenster werden dann die in dieser Datenbank vorhandenen Themen angezeigt. Für ein gewähltes Thema wird dann der Extent angezeigt.

Als nächstes werden von der Datenbank alle Features angefordert, die innerhalb einer Bounding-Box liegen. In dieser einfachen Implementierung wird der Extent des Themas als Bounding-Box verwendet. Deshalb liefert die Anfrage alle Features des Themas zurück. Die zurückgelieferten Features, die innerhalb der Bounding-Box liegen, könnten dann in dem Programm weiterverarbeitet und zum Beispiel auf dem Bildschirm ausgegeben werden.

Für eine Beispieldatenbank, aus der eine Thema mit den Grenzen der Bundesländer von Deutschland ausgewählt wurde, liefert das Programm die folgende Ausgabe:

- Gewähltes Thema: bundeslaender
- Extent des Themas:
 - X min/max: 3280387.0 / 3921542.0
 - Y min/max: 5237512.0 / 6103327.5
- Anzahl der Features innerhalb der Bounding-Box: 16

```
import java.util.HashMap;
import java.util.Map;

import javax.swing.JOptionPane;

import org.geotools.data.DataStore;
import org.geotools.data.DataStoreFinder;
import org.geotools.data.FeatureResults;
import org.geotools.data.FeatureSource;
import org.geotools.feature.FeatureCollection;
import org.geotools.filter.AbstractFilter;
import org.geotools.filter.BBoxExpression;
```

```
import org.geotools.filter.Filter;
import org.geotools.filter.FilterFactory;

import com.vividsolutions.jts.geom.Envelope;

/**
 * Aufbau einer Datenverbindung mit den GeoTools
 * @author Christian Schanz
 */
public class SpearfishPostGIS {

    public static void main(String[] args) throws Exception {

        // Datenbankparameter festlegen
        Map params = new HashMap();
        params.put("dbtype", "postgis");
        params.put("host", "localhost");
        params.put("port", new Integer(5432));
        params.put("database", "tmp");
        params.put("user", "uaie");
        params.put("passwd", "");

        // Verbindung zur Datenbank herstellen
        DataStore pgDatastore = DataStoreFinder.getDataStore(params
        );
        // Parameter zur Optimierung der Datenverbindung
        // pgDatastore.setWKBenabled(true);
        // Für verschiedene Treiberversionen können/müssen
        // verschiedene
        // Parameter gesetzt werden!

        // Auslesen der in der Datenbank verfügbaren Themen und
        // Auswahl eines
        // Themas
        String[] themen = pgDatastore.getTypeNames();
        Object selectedTheme = JOptionPane.showInputDialog(null,
        "Bitte Thema wählen", "PostGIS-Themen",
        JOptionPane.QUESTION_MESSAGE, null, themen, themen[0]);
        System.out.println("- Gewähltes Thema: " + selectedTheme);

        // Bestimmung des Extends des Themas
        FeatureSource featureSource = pgDatastore
        .getFeatureSource((String) selectedTheme);
```

```
Envelope extent = featureSource.getBounds();
System.out.println("-Extent des Themas:");
System.out.println("Xmin/max:" + extent.getMinX() + "/"
    + extent.getMaxX());
System.out.println("Ymin/max:" + extent.getMinY() + "/"
    + extent.getMaxY());

// Auslesen von Features aus der Datenbank
// Zunächst wird die Bounding-Box der zu wählenden Features
// festgelegt.
// Als Beispiel wird der Extent des Themas verwendet. Es
// werden also
// alle Features ausgelesen
Envelope boundingBox = extent;
// Über einen Filter wird die Auswahl der Features
// formuliert
FilterFactory ff = FilterFactory.createFilterFactory();
// Die Features werden über eine Bounding-Box ausgewählt
BBoxExpression bb = ff.createBBoxExpression(boundingBox);
Filter bboxFilter = ff
    .createGeometryFilter(AbstractFilter.GEOMETRY_BBOX);
((org.geotools.filter.GeometryFilter) bboxFilter).
    addRightGeometry(bb);
String geom = pgDatastore.getSchema((String) selectedTheme)
    .getDefaultGeometry().getName();
((org.geotools.filter.GeometryFilter) bboxFilter).
    addLeftGeometry(ff
        .createAttributeExpression(pgDatastore
            .getSchema((String) selectedTheme), geom));

// Der Filter ist formuliert. Jetzt erfolgt die Anfrage an
// die Datenbank
// In featureResult sind dann die Features der Auswahl
// enthalten
FeatureResults featureResults = featureSource.getFeatures(
    bboxFilter);

// Aus dem featureResults Objekt können jetzt die
// gewünschten
// Informationen extrahiert werden,
// hier zum Beispiel die Anzahl der Features
```

```
System.out.println("- Anzahl der Features innerhalb der  
    Bounding-Box: "  
    + featureResults.getCount());  
// Die Features werden an eine FeatureCollection übergeben  
    und können  
    // zum Beispiel  
    // zur Darstellung verwendet werden  
FeatureCollection features = featureResults.collection();  
}  
}
```


Literaturverzeichnis

- [1] ANTONIN GUTTMAN: R-Trees: A Dynamic Index Structure for Spatial Searching. In: BEATRICE YORMARK (Hrsg.): *SIGMOD Conference* Bd. SIGMOD '84, Proceedings of Annual Meeting, Boston, Massachusetts, Juni 18-21, 1984, ACM Press, 47-57
- [2] BARTUNOV, O. ; SIGAEV, T. : *Gevel*. <http://www.sai.msu.su/~megera/oddmuse/index.cgi/Gevel>. Version: 2006-01-20
- [3] BARTUNOV, O. ; SIGAEV, T. : *PostgreSQL and GiST*. http://www.sai.msu.su/~megera/postgres/talks/gist_tutorial.html. Version: 2006-01-20
- [4] BAYER, R. ; MCCREIGHT, E. : Organization an Maintenance of Large Ordered Indexes. In: *Acta Informatica* 1 (1972), Nr. 3, S. 173–189
- [5] BECKMANN, N. ; KRIEGEL, H.-P. ; SCHNEIDER, R. ; SEEGER, B. : The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In: GARCIA-MOLINA, H. (Hrsg.) ; JAGADISH, H. V. (Hrsg.): *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, 322-331
- [6] BRINKHOFF, T. : *Geodatenbanksysteme in Theorie und Praxis - Einführung in objektrelationale Geodatenbanken unter besonderer Berücksichtigung von Oracle Spatial*. Herbert Wichmann Verlag, Heidelberg, 2005
- [7] BUHMANN, E. ; WIESEL, J. : *GIS-Report 2003*. Harzer Verlag <http://www.harzer.de/gr03web.pdf>
- [8] BUHMANN, E. ; WIESEL, J. : *GIS-Report 2004*. Harzer Verlag <http://www.harzer.de/gr04web.pdf>
- [9] CHUCK MURRAY ET AL.: *Oracle Spatial User's Guide and Reference 10g Release 1 (10.1)*. http://www.oracle.com/technology/products/spatial/spatial_doc_index.html. Version: 2006-01-20

- [10] DISY INFORMATIONSSYSTEME GMBH: *disy GISterm*. http://www.disy.net/disy_gisterm.html. Version: 2006-01-20
- [11] E. F. CODD: A Relational Model for Large Shared Data Banks. In: *Communications of the ACM* 13 (1970), Nr. 6, 377-387. <http://www.acm.org/classics/nov95/toc.html>
- [12] EISENTRAUT, P. : *PostgreSQL GE-PACKT*. mitp Verlag, Bonn, 2005
- [13] ENVIRONMENTAL SYSTEMS RESEARCH INSTITUTE (ESRI): *ESRI Shapefile Technical Description*. <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>. Version: 2006-01-20
- [14] ERIC S. RAYMOND: *The Cathedral and the Bazaar, Version 3.0*. <http://www.catb.org/~esr/writings/cathedral-bazaar/>. Version: 2006-01-20
- [15] FREE SOFTWARE FOUNDATION (FSF): *Various Licenses and Comments about them*. <http://www.fsf.org/licensing/licenses/license-list.html>. Version: 03 2005
- [16] FREE SOFTWARE FOUNDATION (FSF): *GPLv3*. <http://gplv3.fsf.org>. Version: 2006-01-20
- [17] GEOTOOLS.ORG: *GeoTools - Feature List*. <http://www.geotools.org/display/GEOTOOLS/Feature+List>. Version: 2006-01-20
- [18] GÖBEL, R. : Effiziente Verwaltung geographischer Daten mit räumlichen Datenbanksystemen - Möglichkeiten und Grenzen. In: STROBL, G. (Hrsg.): *Angewandte Geoinformatik 2005 - Beiträge zum 17. AGIT-Symposium Salzburg* Bd. 17, 2005, S. 823
- [19] HELLERSTEIN, J. M. ; NAUGHTON, J. F. ; PFEFFER, A. : Generalized search trees for database systems. In: *Proceedings of the 21st International Conference on Very Large Data Bases, Zurich, Switzerland* (1995)
- [20] HOEFLE, B. : *Visualising GiST index geometry with PostGIS and GRASS GIS*. http://www.sai.msu.su/~megeera/oddmuse/index.cgi/PosGIS_and_Grass. Version: 2006-01-20
- [21] IBM: *DB2 Universal Database Express Edition*. <http://www-306.ibm.com/software/data/db2/udb/db2express/>. Version: 2006-01-20

- [22] IHM, J. ; LOPEZ, X. : Oracle Locator: Location-Enabling Every Oracle Database - Whitepaper. (2005), 06. http://www.oracle.com/technology/products/spatial/pdf/10gr2_collateral/locator_twp_10gr2.pdf
- [23] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO): *ISO/IEC 9075:1999 Information technology - Database languages - SQL*. 1999
- [24] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO): *ISO/IEC 9075:2003 Information technology - Database languages - SQL*. 2003
- [25] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO): *ISO 19125-1:2004 Geographic Information - Simple Feature Access - Part 1: Common Architecture*. 2004
- [26] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO): *ISO 19125-2:2004 Geographic Information - Simple Feature Access - Part 2: SQL Option*. 2004
- [27] JAEGER, T. ; METZGER, A. : Neues Recht für freie Software – GPLv3 – Gesetzgebung in Vertragsform. In: *c't magazin für computer technik* (2006), Nr. 4/2006, S. 290
- [28] LAUSEN, G. ; VOSSER, G. : *Objekt-orientierte Datenbanken: Modelle und Sprachen*. Oldenbourg-Verlag München, 1996
- [29] LEUTENEGGER, S. ; EDGINGTON, J. ; LOPEZ, M. A.: STR: A Simple and Efficient Algorithm for R-Tree Packing. In: *Proc. 12 th Intern Conf. on Data Engin.*, 1997, S. 497–506
- [30] MICROSOFT: *Shared Source Initiative: Licensing Overview*. <http://www.microsoft.com/resources/sharedsource/default.aspx>. Version: 2006-01-20
- [31] MYSQL AB: *MySQL Licensing Policy*. <http://www.mysql.de/company/legal/licensing/>. Version: 03 2004
- [32] MYSQL AB: *MySQL*. <http://www.mysql.de/>. Version: 2006-01-20
- [33] OPEN GEOSPATIAL CONSORTIUM (OGC): *OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture*, 11 2005. http://portal.opengeospatial.org/files/?artifact_id=13227
- [34] OPEN GEOSPATIAL CONSORTIUM (OGC): *OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SLQ option*, 11 2005. http://portal.opengeospatial.org/files/?artifact_id=13228

- [35] OPEN GEOSPATIAL CONSORTIUM (OGC): *Members - Listed by Name*. <http://www.opengeospatial.org/about/?page=members&view=Name>. Version: 2006-01-20
- [36] OPEN GEOSPATIAL CONSORTIUM (OGC): *OpenGIS Specifications*. <http://www.opengeospatial.org/specs/?page=specs>. Version: 2006-01-20
- [37] OPEN GEOSPATIAL CONSORTIUM (OGC): *Products Compliant to or Implementing OGC Specs or Interfaces*. <http://www.opengeospatial.org/resources/?page=products>. Version: 2006-01-20
- [38] OPEN GIS CONSORTIUM (OGC): *OpenGIS Simple Features Specification For SQL - Revision 1.1*. <http://www.opengeospatial.org/docs/99-049.pdf>. Version: 2006-01-20
- [39] OPEN SOURCE INITIATIVE (OSI): *History of the OSI*. <http://www.opensource.org/docs/history.php>
- [40] OPEN SOURCE INITIATIVE (OSI): *Licensing - The Approved Licenses*. <http://opensource.org/licenses/>. Version: 2005
- [41] OPEN SOURCE INITIATIVE (OSI): *The BSD License*. www.opensource.org/licenses/bsd-license.php. Version: Juli 2006-01-20
- [42] OPEN SOURCE INITIATIVE (OSI): *The Open Source Definition Version 1.9*. <http://www.opensource.org/docs/definition.php>. Version: 11 2006-01-20
- [43] ORACLE: *Oracle Database 10g Express Edition*. <http://www.oracle.com/technology/products/database/xe/index.html>. Version: 2006-01-20
- [44] POSTGRESQL GLOBAL DEVELOPMENT GROUP: *A Brief History of PostgreSQL*. <http://www.postgresql.org/about/history>. Version: 03 2005
- [45] POSTGRESQL GLOBAL DEVELOPMENT GROUP: *PostgreSQL Index Types*. <http://www.postgresql.org/docs/8.1/interactive/indexes-types.html>. Version: 2006-01-20
- [46] REFRACTIONS RESEARCH: *PostGIS Manual*. <http://postgis.refractions.net/docs/>. Version: 2006-01-20
- [47] REFRACTIONS RESEARCH: *Refractions leads PostgreSQL GiST Improvement Project*. <http://refractions.net/news/index.php?file=20051003.data>. Version: 2006-01-20

- [48] REFRACTIONS RESEARCH: *Refractions Research*. <http://www.refractions.net/>.
Version: 2006-01-20
- [49] RENNER, T. ; VETTER, M. ; REX, S. ; KETT, H. : Open Source Software: Einsatzpotenziale und Wirtschaftlichkeit / Fraunhofer Gesellschaft, Competence Center Electronic Business. 2005. – Forschungsbericht
- [50] RONNEBURG, F. : *Debian Anwenderhandbuch - Die Definition quelloffener Software*. <http://debiananwenderhandbuch.de/freiesoftware.html>. Version: 2006-01-20
- [51] SCHEUGENPFLUG, S. J.: *Relationale und Objektrelationale Datenbankkonzepte in Geoinformationssystemen*, Technische Universität München, Institut für Geodäsie, GIS und Landmanagement, Fachgebiet Geoinformationssysteme, Diss., 2005. <http://www.gis1.bv.tum.de/Forschung/Promotionen/Dokumente/Scheugenpflu%g%20200506.pdf>. – Elektronische Ressource
- [52] SCHILCHER, M. : Geoinformationssysteme – Zwischenbilanz einer stürmischen Entwicklung. In: *Zeitschrift für Vermessungswesen (ZfV)* 121 (1996), Nr. 8, 361-377. <http://www.gis.bv.tum.de/public/schilcher/zfv-artikel/zfv.htm>
- [53] SELLIS, T. ; ROUSSOPOULOS, N. ; FALOUTSOS, C. : The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In: *Proc. International Conference on Very Large Data Bases*, 507-518
- [54] SNYDER, B. : *Open Source Landscape Series - Open Source Relational Databases and the Enterprise*. <http://virtuas.com/osl-osrdb-01.pdf>. Version: 07 2005
- [55] STALLMAN, R. ; GAY, J. (Hrsg.): *Free Software - Free Society*. GNU press <http://www.gnu.org/philosophy/fsfs/rms-essays.pdf>
- [56] STALLMAN, R. : *The GNU Project*. <http://www.gnu.org/gnu/thegnuproject>.
Version: 2006-01-20
- [57] STALLMAN, R. ; MOGLEN, E. : *GNU General Public License, Version 2*. <http://www.fsf.org/licensing/licenses/gpl.html>. Version: 06 1991
- [58] SUN MICROSYSTEMS: *JDBC Technology*. <http://java.sun.com/products/jdbc/>.
Version: 2006-01-20
- [59] THE APACHE SOFTWARE FOUNDATION: *The Apache Software Foundation*. <http://www.apache.org>. Version: 2006-01-20

- [60] THE ECLIPSE FOUNDATION: *The Eclipse Foundation*. <http://www.eclipse.org>.
Version: 2006-01-20
- [61] TORVALDS, L. ; DIAMOND, D. : *Just for Fun - Wie ein Freak die Computerwelt revolutionierte*. Hanser, 2001
- [62] VIVID SOLUTIONS: *Vivid Solutions Inc.* <http://vivid solutions.com/>.
Version: 2006-01-20
- [63] WIKIPEDIA: *GNU General Public License*. http://de.wikipedia.org/wiki/GNU_General_Public_License. Version: 2006-01-20
- [64] WILLIAMS, S. : *Free as in Freedom - Richard Stallman's Crusade for Free Software*. O'Reilly <http://www.oreilly.com/openbook/freedom/>
- [65] ZIEGLER, M. : *Untersuchung geographischer Anfragesprachen auf der Basis relationaler und objektrelationaler Datenbankmanagementsysteme*, Technische Universität München, Geodätisches Institut, Fachgebiet Geoinformationssysteme, Diss., 2002. http://www.gis1.bv.tum.de/Forschung/Promotionen/Dokumente/Ziegler_2002.pdf. – Elektronische Ressource