CityGML in PostGIS –

Portierung, Anwendung und Performanz-Analyse am Beispiel der 3DCityDB Berlin

von Felix Kunde

Masterarbeit zur Erlangung des Akademischen Grades

Master of Science (M. Sc.)

im Studiengang Geoinformation und Visualisierung
am Institut für Geographie im Fachbereich Geoinformatik
der Universität Potsdam

Erstgutachter: Prof. Dr. Hartmut Asche (Universität Potsdam)
Zweitgutachter: Dipl. Inform. Gerhard König (Technische Universität Berlin)

Dieses Werk ist unter einem Creative Commons Lizenzvertrag lizenziert:
Namensnennung - Keine kommerzielle Nutzung - Weitergabe unter gleichen Bedingungen 3.0 Unported
Um die Bedingungen der Lizenz einzusehen, folgen Sie bitte dem Hyperlink:
http://creativecommons.org/licenses/by-nc-sa/3.0/

Online veröffentlicht auf dem
Publikationsserver der Universität Potsdam:
URL http://opus.kobv.de/ubp/volltexte/2013/6365/
URN urn:nbn:de:kobv:517-opus-63656
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-63656

Zusammenfassung

Der internationale Standard CityGML ist zu einer zentralen Schnittstelle für die geometrische wie semantische Beschreibung von 3D-Stadtmodellen geworden. Das Institut für Geodäsie und Geoinformationstechnik (IGG) der Technischen Universität Berlin leistet mit ihren Entwicklung der *3D City Database* und der *Importer/Exporter* Software einen entscheidenden Beitrag die Komplexität von CityGML-Daten in einer Geodatenbank intuitiv und effizient nutzen zu können. Die Software des IGG ist Open Source, unterstützte mit *Oracle Spatial* (ab Version 10g) aber bisher nur ein proprietäres Datenbank Management System (DBMS).

Im Rahmen dieser Masterarbeit wurde eine Portierung auf die freie Datenbank-Software *PostgreSQL/PostGIS* vorgenommen und mit der Performanz der *Oracle*-Version verglichen. *PostGIS* gilt als eine der ausgereiftesten Geodatenbanken und wurde in diesem Jahr mit dem Release der Version 2.0 nochmals um zahlreiche Funktionen und Features (u.a. auch 3D-Unterstützung) erweitert. Die Ergebnisse des Vergleiches sowie die umfangreiche Gegenüberstellung aller verwendeten Konzepte (SQL, PL, Java) geben Aufschluss auf die Charakteristika beider räumlicher DBMS und ermöglichen einen Erkenntnisgewinn über die Projektgrenzen hinaus.

Abstract

The international standard CityGML has become a key interface for describing 3D city models in a geometric and semantic manner. With the relational database schema *3D City Database* and an according *Importer/Exporter* tool the Institute for Geodesy and Geoinformation (IGG) of the Technische Universität Berlin plays a leading role in developing concepts and tools that help to facilitate the understanding and handling of the complex CityGML data model. The software itself runs under the Open Source label but yet the only supported database management system (DBMS) is *Oracle Spatial* (since version 10g), which is proprietary.

Within this Master's thesis the *3D City Database* and the *Importer/Exporter* were ported to the free DBMS *PostgreSQL/PostGIS* and compared to the performance of the *Oracle* version. *PostGIS* is one the most sophisticated spatial database systems and was recently extended by several features (like 3D support) for the release of the version 2.0. The results of the comparison analysis as well as a detailed explanation of concepts and implementations (SQL, PL, Java) will provide insights in the characteristics of the two DBMS that go beyond the project focus.

Erklärung

Ich versichere hiermit, dass die vorliegende Masterarbeit selbstständig verfasst wurde. Es wurden keine weiteren als die angegebenen Quellen und Hilfsmittel benutzt. Die Stellen der Arbeit, die in anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, wurden durch Angaben der Quellen im Text gekennzeichnet.

Felix Kunde

Berlin, den 27.08.2012

Danksagung

Ich möchte mich an dieser Stelle bei allen Personen bedanken, die mich während der letzten Monate bei der Umsetzung der Arbeit mit Tat und Rat unterstützt haben.

An erster Stelle danke ich meinem Erstgutachter Prof. Dr. Hartmut Asche für wegweisende Gespräche zur Findung eines Themas und für die Flexibilität zu einer Kooperationsarbeit mit der TU-Berlin.

Ich danke Gert König, dass er sich meinen Interessen für eine Masterarbeit im Bereich der Geodatenbanken angenommen hat und entscheidende Schritte mit den Entwicklern der *3D City Database* in die Wege geleitet hat. Ich danke ihm für die Rücksprachen zu den Forstschritten meiner Arbeit.

Mein größter Dank gebührt Javier Herreruela und Claus Nagel für die tolle Zusammenarbeit bei der Software-Entwicklung und -Verwaltung. Ihre Türen standen immer offen für Ratschläge und gemeinsame Diskussion. Jedes Treffen brachte neue Erkenntnisse und gab Antrieb zur Weiterentwicklung.

Ich bedanke mich auch bei Prof. Dr. Thomas Heinrich Kolbe für sein reges Interesse an der Masterarbeit und den vielen Denkanstößen bei gemeinsamen Gesprächen.

Dr. Lutz Ross und Ingolf Jung von der virtualcitySYSTEMS GmbH ermöglichten mir parallel zu der Arbeit ein spannendes Praktikum zu absolvieren, in dessen Rahmen sie mir auch Freiräume ließen, die *PostGIS*-Portierung zu einer vollwertigen Version zu entwickeln. Ich bedanke mich auch bei allen restlichen Mitarbeitern von virtualcitySYSTEMS für das sehr angenehme Arbeitsklima.

Laure Fraysse danke ich für die geleisteten Vorarbeiten zur *PostGIS*-Umsetzung der *3D City Database* und für die Bereitstellung ihres Praktikumsberichtes bei der Firma IGO. Ihre Ansätze halfen mir mich gut in den praktischen Teil der Arbeit einzufinden.

Während der Entwicklungszeit sind zahlreiche Kontakte zu Vertretern aus Wirtschaft und Forschung entstanden, die durch ihr Interesse an dem Masterprojekt die eigene Motivation stets aufs Neue beflügelten.

Gute Freunde sorgten immer wieder für eine dringende Abwechslung vom Arbeitsalltag. Danke Nikias, Jakob, Fabi, Thorben, Jessica, Tine, Malte, Lukas, David. Danke auch an meine Mitbewohner Johannes, Annika, Sarah und Steve für ein tolles WG-Leben und sorry, wenn ich im letzten Jahr allzu oft am Rechner klebte. Und danke Ramona für die vielen Gespräche und die schöne gemeinsame Zeit im und neben dem Studium.

Zuletzt danke ich meinen Eltern für die Unterstützung aus der Ferne und meinen Brüdern für den Austausch von Gedanken abseits von Arbeit und Berlin.

Inhaltsverzeichnis

Prolog

Kapitel 1 - Einleitung....1
1.1. Der Open-Source-Gedanke als Motivator der Arbeit....2
1.2. Aufbau der Arbeit....2
1.3. Fragestellungen und Zielsetzung....3
1.4. Datengrundlage, verwendete Soft- und Hardware 3

Teil I - Einführung in die Thematik

Kapitel 2 - Ein kurzer Überblick zu 3D-Stadtmodellen....7
2.1. Von geschlossenen Systemen zum interoperabelen Modell....7
2.2. Das Konzept der LODs....8
2.3. Wie werden Daten erhoben?....9
2.3.1. Photogrammetrische Verfahren....9
2.3.2. Prozedurale Verfahren....11
2.3.3. Detailmodellierung....11
2.4. Anwendungsgebiete für 3D-Stadtmodelle....12
2.4.1. Einbettung in einer GDI....12

Kapitel 3 - CityGML....13
3.1. Aufbau von CityGML....14
3.2. Das geometrische Modell von CityGML....15
3.3. Vom Datenmodell zum relationalen Datenbankschema....15
3.4. Aktuelle Entwicklung....18
3.5. Softwareunterstützung für CityGML....18
3.6. Vergleich zu anderen 3D-Formaten....20

Kapitel 4 - Wahl der Datenbank: Relationale DBMS....21
4.1. Oracle 11g mit Spatial Extension....22
4.2. PostgreSQL 9.1....22
4.3. PostGIS 2.0....23
4.4. Andere RDBMS mit räumlichen Datentypen....24
4.5. NoSQL - Alternativen für die Speicherung von CityGML-Dateien?....25
4.6. Verwandte Arbeiten....26

Teil II - Die Portierung der 3DCityDB

Kapitel 5 - Anforderungsprofil für die PostGIS-Version der 3DCityDB....29
5.1. Bisheriger Aufbau der 3DCityDB in Oracle....29
5.2. Notwendige Umsetzungen im Rahmen der Masterarbeit....31

Kapitel 6 - Die Portierung auf PostgreSQL/PostGIS.....................................33
6.1. Anpassungen der SQL-Skripte für das Datenbankschema.......................................33
6.1.1. Unterschiede bei den Datentypen...33
6.1.2. Constraints und Trigger..38
6.1.3. Unterschiede bei räumlichen Indizes...38
6.1.4. CREATE_DB.sql..39
6.2. Übersetzung der PL/SQL-Skripte...39
6.2.1. PL/SQL Packages versus PostgreSQL Schemata...41
6.2.2. Abfragen von Systemtabellen...43
6.3. Umprogrammierung des Java-Clients Importer/Exporter..43
6.3.1. Wie arbeitet der Importer/Exporter?..44
6.3.2. Projektaufbau in der Entwicklungsumgebung Eclipse.......................................46
6.3.3. Die wichtigsten Änderungen im Quellcode..47
6.3.4. Zukünftige Struktur der Software...50

Teil III – Performanz-Vergleich beider Versionen

Kapitel 7 - Konfiguration und Performance-Monitoring................................53
7.1. Best Practices...53
7.2. Konfiguration des DBMS..54
7.2.1. Das sich selbst verwaltende Oracle..54
7.2.2. postgresql.conf..55
7.3. Überwachungsmethoden der DBMS-Performanz...56
7.3.1. Versuchsaufbau..56
7.3.2. Datenbank- und System-Monitoring..58

Kapitel 8 - Ergebnisse des Performanz-Analyse................................59
8.1. Import von Datensätzen..60
8.2. Export von Datensätzen...64
8.3. Speicherbedarf...67
8.4. Rechnerauslastung...68

Kapitel 9 - Bewertung der Ergebnisse...69
9.1. Interpretation der Ergebnisse...69
9.1.1. Welches DBMS war schneller?...70
9.1.2. Welches DBMS skalierte besser?..71
9.2. Reflexion zu geleisteten und zukünftigen Portierungen..72
9.3. Alternativen für nicht direkt portierbare Komponenten..73

Epilog

Kapitel 10 - Fazit...75

Verzeichnisse..77
Literaturverzeichnis...77
Internetquellen..87

Abbildungsverzeichnis....89
Tabellenverzeichnis....91
Fremdwörterverzeichnis....92
Abkürzungsverzeichnis....94

Anhang

Anhang A – SQL und PL/pgSQL Skripte....99
A.1. Toplevel-Dateien....99
A.2. Schema-Ordner....109
A.3. Hilfsskripte....139
A.4. PL/pgSQL-Skripte....142

Anhang B – Java Quellcode....241
B.1. Die Verbindung zur Datenbank....242
B.2. Aufruf von PL/pgSQL Funktionen....244
B.2.1. Funktionen aus den Dateien IDX, UTIL und STAT....244
B.2.2. Das Berechnen der Bounding Box....245
B.3. Datenbank-Spezifika in Java....247
B.3.1. Das Datenbank-SRS....247
B.3.2. Bounding Box Filter und Optimizer Hints....248
B.3.3. Abfragen für den Import....249
B.3.4. Die „nologging“ Option....250
B.3.5. Datentypen in Cache Tabellen....250
B.4. Implizite Sequenzen....251
B.5. Die Verarbeitung von Geometrien....252
B.5.1. Von CityGML zur 3DCityDB....252
B.5.2. Von der 3DCityDB zu CityGML....256
B.6. Die Verarbeitung von Texturen und binären Daten....262
B.6.1. Der Import von Texturen und binären Daten....263
B.6.2. Der Export von Texturen und binären Daten....266
B.7. Die Batchsize von PostgreSQL....268
B.8. Workspace Management....269
B.9. KML-Exporter....270
B.9.1. Datenbank-Abfragen....270
B.9.2. Geometrien für KML Placemarks....275
B.9.3. Texturen beim COLLADA-Export....279

Anhang C – Dokumentation der Testergebnisse....281

Anhang D – Weitere Abbildungen....285
D.1. Screenshots vom Importer/Exporter....285
D.2. Screenshots von verwendeten Stadtmodelldaten....291
D.3. Sonstige Abbildungen....295

Anhang E – 3D City Database 2.0.6-postgis Tutorial....297

Prolog

Kapitel 1

Einleitung

Nach fast zehn Jahren Entwicklung ist CityGML heute ein weltweit anerkannter und vom Open Geospatial Consortium (OGC) offiziell festgeschriebener Standard für die geometrische wie semantische Beschreibung von virtuellen 3D-Stadt- und Landschaftsmodellen. Er dient dazu die Zusammenarbeit von unterschiedlichen Nutzern aus Stadtplanung, Wirtschaft und Forschung flexibler zu gestalten und lässt sich als Ausgangsbasis für neue Projekte verwenden und erweitern [vgl. Kolbe 2009, www1, www2].

OGC-Standards wie CityGML sind in der Geoinformatik allgegenwärtig. Sie erweitern die Operationalität von monolithischer Software und service-orientierten Architekturen (SOA) gleichermaßen, beschleunigen den Datentransfer und verbessern die Planung und Administration von digitalen Infrastrukturen. Dieses Erfolgsrezept gelingt in der Praxis jedoch nur mit zuverlässigen Werkzeugen.

Den Architekturen klassischer Geoinformationssysteme (GIS) (Erfassung, Verwaltung, Analyse und Präsentation [vgl. Bill 2010]) mangelt es – historisch bedingt – an einer strikten Trennung der Daten zur Anwendung. Bei der stetig wachsenden Menge an digitalen Daten mit Raumbezug ist eine dauerhaft flexible Lösung notwendig [Bartelme 2005: 301; Brinkhoff 2008: 5]. Datenbank Management Systeme (DBMS) können dies gewährleisten und sind mittlerweile auch bei der Speicherung von Geodaten so ausgreift, dass sie herkömmliche GI-Systeme als zentrale Datenhaltungs- und Analysekomponente ablösen. Für die Verwaltung des Berliner Stadtmodells in CityGML wurde an der Universität Bonn und der Technischen Universität Berlin mit der *3D City Database* ein Schema entwickelt, das auf einem objekt-relationalen Geodatenbanksystem aufsetzt. Die virtuellen Modelle können dadurch konsistent wie persistent gespeichert werden, was wiederum zeiteffiziente komplexe Raumanalysen ermöglicht. Mit dem von der TU-Berlin entwickelten Konverter *Importer/Exporter* können die Objekte aus CityGML-Dateien in die Tabellen der *3D City Database*, in der Folge mit *3DCityDB* abgekürzt, abgebildet werden.

1.1. Der Open-Source-Gedanke als Motivator der Arbeit

Die *3DCityDB* und der *Importer/Exporter* wurden unter der GNU Lesser General Public Licence v3 (LGPL) veröffentlicht und sind frei im Internet verfügbar [vgl. Kolbe et al. 2009, www3]. Die Anforderungen, welche im Jahre 2007 vor der Weiterentwicklung einer *3DCityDB*-Version für CityGML 1.0.0 definiert wurden, konnten nur durch *Oracle Spatial* (ab Version *10g Release 2*) vollständig abgedeckt werden. Im weiteren Projektverlauf wurde daher ausschließlich dieses DBMS unterstützt. Bei *Oracle Database* handelt es sich um eine proprietäre Datenbank-Software, die zwar kostenlos heruntergeladen und verwendet werden kann, in einem kommerziellen Umfeld jedoch gegen Gebühren lizenziert werden muss. Oracle bietet verschiedene Varianten seiner Datenbank an, die sich je nach Komplexität im Preis unterscheiden. Die *Spatial* Erweiterung gibt es nur für die *Enterprise Edition* (EE), dem teuersten Produkt der Firma. In einem Beispiel aus [Ahrends 2011b: 4] wird schon bei drei Server-Lizenzen mit *Oracle-EE* fast die Million in Dollar überschritten. Natürlich ist der darin eingeschlossene Support seitens Oracle zu berücksichtigen, für kleine bis mittelständische Unternehmen ist der finanzielle Aufwand hingegen kaum zu stemmen.

Die ersten Entwicklungen der Uni Bonn unterstützten noch *PostGIS*, die räumliche Erweiterung des Open-Source-DBMS *PostgreSQL*, welche *Oracle Spatial* funktional am nächsten kommt. *PostGIS* wurde mit dem Release einer 2.0 Version in diesem Jahr in den Bereichen 3D-Geometrie und Rasterdaten umfassend erweitert. Damit verfügt es über genug Potential, um eine in den letzten Jahren angedachte Portierung der bisherigen *3DCityDB*-Softwarelösungen zu wagen. Langfristig gesehen könnte ein komplett auf Open-Source-Bestandteilen basierendes Konzept nicht nur die Kosten reduzieren, sondern auch zu einem breiteren Einsatz und steigenden Nutzen von CityGML beitragen. In Vermessungs- und Geoinformationseinrichtungen des Öffentlichen Dienstes bildet CityGML bereits den Konsens, wenn es um die Verwaltung von dreidimensionale Bestandsdaten geht. Seit Ende 2009 schreibt die Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der BRD (AdV) die landesweite Erhebung und Haltung von 3D-Modellen vor [vgl. GeoInfoDok 2009], woraus sich auch auf längere Sicht ein Bedarf für eine *PostGIS*-Version der *3DCityDB* generiert.

1.2. Aufbau der Arbeit

Die Arbeit gliedert sich in drei Teile, die wiederum einzelne Kapitel enthalten:

- In Teil 1 werden die in der Einleitung erwähnten Inhalte vertieft und der thematische Rahmen aufgezogen, in dem sich diese Arbeit bewegt. Der Aufbau und die Entwicklung des CityGML-Datenmodells wird im Detail besprochen und seine Anwendung gegen mehrere Softwarelösungen evaluiert.

- Teil 2 erläutert zunächst den Aufbau der *3DCityDB* und skizziert ein Anforderungsprofil für die Umsetzung nach *PostgreSQL/PostGIS*, die dann methodisch ausführlich beschrieben wird.

- In Teil 3 wird die Performanz zwischen der bisherigen *Oracle*-Lösung und der *PostgreSQL/PostGIS*-Adaption verglichen. Alle gewonnenen Erkenntnisse der Arbeit werden danach zusammenfassend in Hinblick auf die Ausgangs-fragestellungen diskutiert. Dabei wird auch die Art der Durchführung der Arbeit reflektiert, um in einem abschließenden Ausblick und Fazit eine Orientierung für zukünftige Projekte zu liefern.

1.3. Fragestellungen und Zielsetzung

Die vorliegende Masterarbeit realisiert einen in Kapitel 5 definierten Rahmen der Umsetzung nach *PostgreSQL/PostGIS*. Ihr Schwerpunkt orientiert sich stärker an praktischen Beispielen als an Theorien, dennoch soll neben operationellen Fragen der Portierung auch zu folgenden konzeptionellen Aspekten ausreichend Stellung genommen werden:

- Können anhand der Ergebnisse der Teilumsetzung Aussagen zum Implementierungsaufwand und Funktionsumfang nicht portierter Software-bausteine getroffen werden?
- Wie stark würden nicht zu realisierende Bestandteile für *PostgreSQL/PostGIS* den professionellen Einsatz der *3DCityDB* einschränken?
- Lassen sich aus den Ergebnissen der Performanz-Analyse, d.h. das Zusammenspiel aus Durchführungszeit, Speicherbedarf und Rechnerauslastung, Charakteristika für beide DBMS postulieren? Können Rückschlüsse zur Effektivität der jeweils verwendeten Datenbankspezifika gezogen werden?

1.4. Datengrundlage, verwendete Soft- und Hardware

Datengrundlage (siehe auch Seite 56 und 57):

- Berliner Stadtmodell:
  - 21,1 GB,
  - 534357 Gebäude,
  - 590746 Adressen,
  - 3935315 Texturen,
  - 9083266 Geometrien

Skripte und Quellcode:

- *3DCityDB 2.0.6*
- *Importer/Exporter 1.4*

RDBMS:

- *Oracle 11gR2 11.2.0.2.0 Enterprise Edition* 64-Bit
- *PostgreSQL 9.1.3* 32-Bit mit *PostGIS 2.0-beta* 32-Bit
- *PostgreSQL 9.1.3* 64-Bit mit *PostGIS 2.0.1.* 64-Bit für abschließende Tests

Administrationswerkzeuge der RDBMS:

- für *Oracle*
  - *SQL*Plus* (Kommandozeile)
  - *SQL Developer* (grafische Oberfläche)
- für *PostgreSQL*
  - *psql* (Kommandozeile)
  - *pgAdminIII* (grafische Oberfläche)

Java-Entwicklungsumgebung:

- *Eclipse 3.7 Indigo*
- *Tigris Subversion* 4.8.x
- *Ant*

Verwendete CPU:

- *3DCityDB*-Server: Intel Xeon HexaCore, Win7 64-Bit, 24 GB RAM
- Test-Rechner: Intel Core2 QuadCore, Win7 64-Bit, 8 GB RAM

Universtäts-interne Netzwerkleitung:

- 100 Mbps

Teil I

Einführung in die Thematik

Was erwartet den Leser?

Kapitel 2 umreißt die Hintergründe zur 3D-Stadtmodellierung. Nach der Historie wird das Konzept der Level-Of-Detail und verschiedene Methoden zur Datenerfassung vorgestellt.

Kapitel 3 geht detaillierter auf CityGML und die bisherigen Implementierungsstrategien sowie auf externe Programme und Services ein, in denen CityGML Verwendung findet.

Kapitel 4 stellt die in der Masterarbeit zum Einsatz gekommenen Relationalen DBMS *Oracle 11g* und *PostgreSQL 9.1* mit ihren jeweiligen räumlichen Erweiterungen vor und zeigt außerdem mögliche Alternativen zur Speicherung von CityGML-Daten auf.

Kapitel 2

Ein kurzer Überblick zu 3D-Stadtmodellen

Die dritte Dimension existiert im Umfeld der Geodaten bereits seit Jahrzehnten in Form der z-Koordinate. Sie bildet die Grundlage für digitale Geländemodelle (DGM) mit dessen Hilfe schon viele geowissenschaftliche Forschungsarbeiten Höhenabhängigkeiten analysierten. Mit dem Aufkommen der ersten Animationsfilme und 3D-Videospiele Mitte der 1990er Jahre wurden auch in der Fernerkundung Versuche unternommen Realweltobjekte dreidimensional abzubilden [vgl. Grün et al. 1995]. Eine fortschreitende Automatisierung bei Luftbildkorrelation und Laserscanning ermöglichte schon wenige Jahre später erste virtuelle Häuserlandschaften [vgl. Brenner, Haala 1998; Baillard et al. 1998]. Doch dem ersten Hype folgte aufgrund von Systembeschränkungen und mangelnde Wiederverwendbarkeit schnell Ernüchterung [vgl. Bauer, Mohl 2005], die letztendlich den Anstoß für CityGML gab (siehe 2.1.). Im weiteren Verlauf dieses Kapitels werden Konzepte und Anwendungsgebiete von 3D-Stadmodellen vorgestellt und es wird erklärt, wie die dazugehörigen Datengrundlagen produziert werden.

2.1. Von geschlossenen Systemen zum interoperablen Modell

Die ersten flächendeckenden Stadtmodelle entstanden um die Jahrtausendwende u.a. in Zürich, Glasgow, Graz, Los Angeles, Stuttgart [vgl. Köninger, Bartel 1998]. Zu Anfang stellten diese Modelle allerdings oft eigenständige Lösungen in geschlossenen Systemen dar und waren zunächst vorrangig an der Visualisierung orientiert [Kolbe 2004: 1]. Die spätere Popularität von *Google Earth* als Darstellungsplattform von 3D-Stadtmodellen verdeutlichte zwar das Gewicht der visuellen Ebene für den Nutzer, eine mangelnde Wiederverwendbarkeit und ineffiziente Bestandsfortführung stellten jedoch die Wirtschaftlichkeit der Modelle in Frage [vgl. Krämer, Hopf 2012; Gröger et al. 2005].

Die Standardisierungsbestrebungen der OGC rund um Web-Mapping-Services (WMS) schlugen die ersten Brücken für einen Informationsaustausch über virtuelle Städte

zwischen unterschiedlichen Infrastrukturen [vgl. Kolbe 2004] und machten die Notwendigkeit nach einem einheitlichen Austauschformat für die Beschreibung von Stadtmodellen immer deutlicher. Zu diesem Zweck gründete sich 2002 die Special Interest Group 3D (SIG 3D) aus der Initiative *Geodateninfrastruktur Nordrhein Westfalen* (GDI NRW), ein Gremium mit Vertretern aus Wirtschaft, Wissenschaft und Öffentlichen Dienst [www4]. Gemeinsam erörterte man die Anforderungen und den Umfang eines Datenformats, dass auf bisherigen OGC-, ISO- und W3C-Standards aufbauen und sie ergänzen sollte (siehe Kapitel 3) [vgl. Herring 2001]. In Anlehnung an dem zugrundeliegenden Standard GML wurde das entwickelte Datenmodell als CityGML bezeichnet.

2.2. Das Konzept der LODs

Angesichts von einst limitierter Performanz beim Echtzeit-Rendering auf Desktop- und ersten Internetanwendungen wurde eine Untergliederung von Stadtmodellen in unterschiedlich detaillierte Darstellungsebenen diskutiert [vgl. Coors, Flick 1998; Köninger, Bartel 1998; Schilcher et al. 1999], so genannte „Level of Detail" (LOD), nach einem Konzept aus der Computergrafik [vgl. Clark 1976, Hoppe 1996]. Aufgrund technischer Fortschritte entspricht die Unterteilung in LOD-Stufen aber mittlerweile ausschließlich einer rein inhaltlichen Konzeption.

Die LOD-Stufen nach [Gröger et al. 2005: 5ff.] am Beispiel von Gebäuden:

- **LOD 0** wird nur durch ein DGM beschrieben, welches ggf. mit Karten oder Satelliten- bzw. Luftbildern versehen werden kann. Es heißt deshalb Regionalmodell.
- In **LOD 1** bilden alle Gebäudeteile eine Einheit, repräsentiert durch eine Volumengeometrie. Die Angabe von Materialeigenschaften für verschiedene Erscheinungen der Häuser ist möglich. Da keine Dachformen existieren wird dieses Basismodell oft auch als „Klötzchenmodell" bezeichnet [Gerschwitz et al. 2011: 293].
- In **LOD 2** besitzen einzelne Gebäudebegrenzungsflächen wie Dächer oder Wände ihre eigene Geometrie, wodurch z.B. Dachüberstände generiert werden können. Die semantische Untergliederung ermöglicht wiederum das Belegen der Einzelflächen mit Objekteigenschaften und lässt so eine spezifische Texturierung und eine Erweiterungen um Merkmale wie Gauben, Balkone und sogar Antennen zu. Das verleiht den Stadtmodellen bereits ein realistisch wirkendes Aussehen.
- Gesteigert werden kann dies in **LOD 3** mit der Modellierung von detaillierten Hausfassaden samt Fenstern und Türen (Öffnungen), alternativ als Architektur-modell beschrieben.
- **LOD 4** erweitert das Schema um den inneren Aufbau der Gebäude. Räume werden durch Volumengeometrien beschrieben, Einrichtungsgegenstände besitzen eigene Objektgeometrien. Es ergibt sich eine große Schnittmenge zu Gebäude-informationssystemen aus dem Facilitymanagement-Bereich (siehe auch 2.3.3.).
- LOD 0 und LOD 4 sind im Übrigen Ergänzungen der SIG 3D.

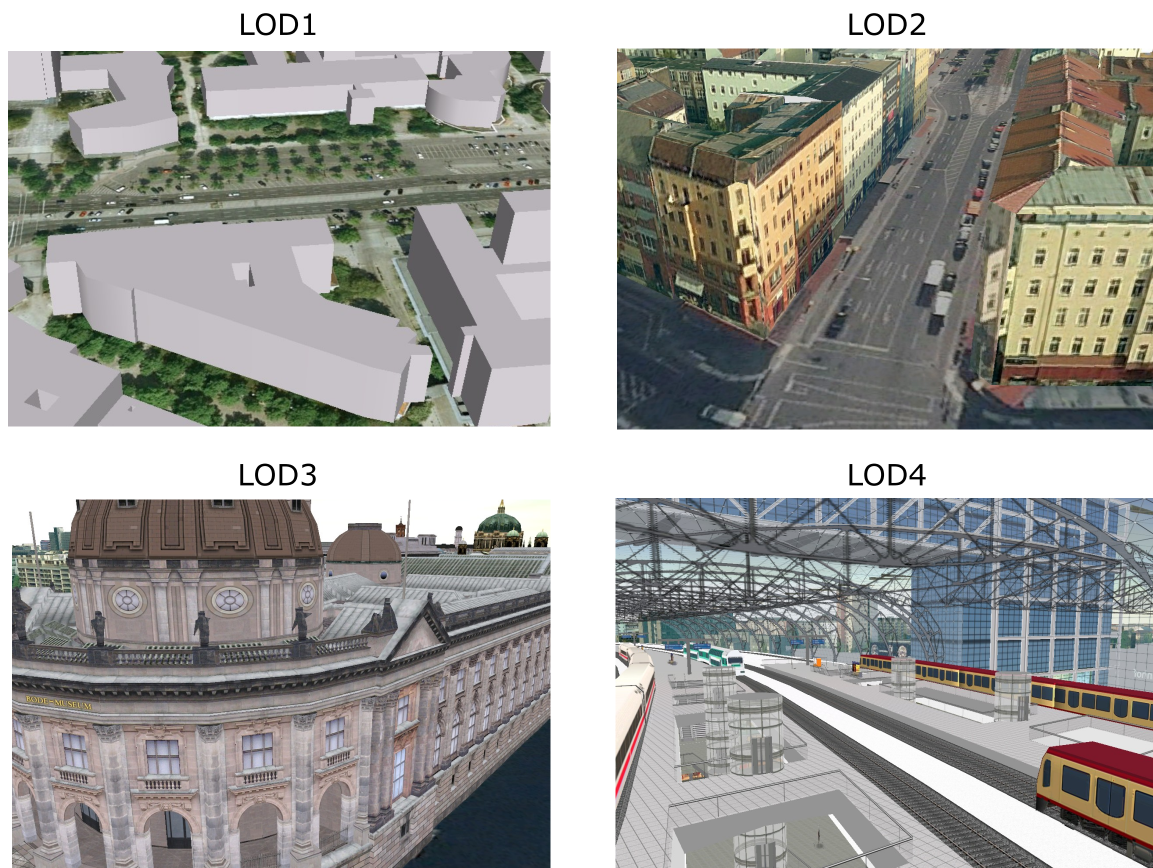


Abb.1: Die LOD-Stufen am Beispiel von Gebäuden des Berliner Stadtmodells

2.3. Wie werden Daten erhoben?

Für den Aufbau von virtuellen Stadtmodellen haben sich die Methoden über die Jahre kaum verändert. In diesem Unterkapitel werden hauptsächlich photogrammetrische und prozedurale Verfahren vorgestellt, die sich für die flächendeckende dreidimensionale Erfassung und Modellierung von Städten bis zu einem Detailgrad von LOD 2 eignen. Verfahren für höhere LOD-Stufen werden nur kurz in 2.3.3. angesprochen.

2.3.1. Photogrammetrische Verfahren

LIDAR-Daten ("Light detection and ranging") aus flugzeuggestützten („Airborne") Laserscanning-Befliegungen sind in den letzten Jahren zu einer bevorzugten Datengrundlage für die Generierung von Digitalen-Höhen-Modellen (DEM) und daraus abzuleitenden 3D-Stadtmodellen geworden [Li, Guan 2011: 76], weil selbst dichte und komplexe Bebauung sehr genau vermessen werden kann [Haala 2010: 570]. Filtermöglichkeiten für reflektierte Laserpulse erlauben sogar Bodenaufnahmen bei dichter Vegetation [Lemmens 2011: 156]. Je nach gewünschtem Detailgrad für das Endprodukt kann die Anzahl der Scanpunkte pro Quadratmeter variiert werden. Für Gebäude sind bereits wenige Punkte pro m² (2 – 8) ausreichend.

In der Literatur wird LIDAR oft als eigenes Anwendungsfeld gegen die klassische Photogrammetrie, d.h. Auswertung von Luftbildern, abgegrenzt. Ein Verfahren dieser klassischen Photogrammetrie ist die Bildkorrelation aus Stereo-Luftbild-Aufnahmen. Dabei werden Bildpixel mit Höhen aus Stereomessungen korreliert und daraus entstehende Objekte mit einem DGM verschnitten [Gülich 2005: 17ff.]. Die Fehlerrate bei abgeleiteten Geometrien ist u.a. bedingt durch Wetterverhältnisse oder sich bewegende Schatten höher als bei LIDAR-Daten. Fortschritte bei den Kamerasystemen, ein gesunkener Aufwand bei der Erstellung von Geländemodellen und nicht zuletzt der niedrigere Preis für Rohdaten gegenüber LIDAR haben zu einer Renaissance der Bildkorrelation geführt [vgl. Haala 2009].

In den letzten Jahren wurde Algorithmen und Programme entwickelt, um aus den erhaltenen Punktwolken von LIDAR oder Bildkorrelation einfache geometrische Modelle größtenteils automatisch ableiten zu können [www5, www6, www7, www8]. Detaillierte Dokumentationen zur technischen Umsetzung finden sich in [Dorninger, Pfeifer 2008] und [Li, Guan 2011].

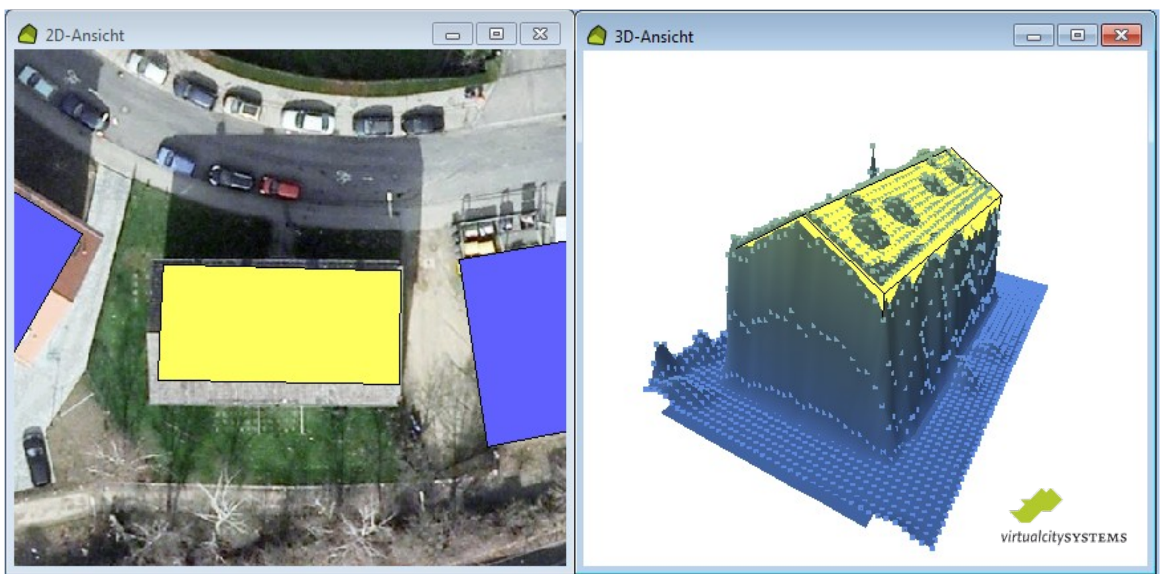


Abb. 2: Automatisierte Ableitung von Gebäuden aus einem DEM (Auschnitt von BuidingReconstruction, einer Software von virtualcitySYSTEMS)

Durch Verschneidung mit Hausumringen aus Vermessungskatastern können die Gebäude grundrisstreu positioniert werden. Durch eine solche Integration mit zweidimensionalen Geodatenbeständen können die 3D-Modelle auch semantisch angereichert werden. Der inhaltliche Mehrwert hängt dabei natürlich von der Qualität der hinzugezogenen Datengrundlage ab.

Trotz beachtlicher Automatisierung ist der Aufwand der manuellen (Nach-)Bearbeitung bei photogrammetrischen Verfahren dennoch sehr hoch. Eine regelmäßige Laserscanning-Befliegung, um die Daten aktuell zu halten, ist nur wirtschaftlich, wenn die Punktwolken zuerst nach Veränderungen klassifiziert werden, um die Rekonstruktion einzugrenzen [vgl. Richter, Döllner 2011].

2.3.2. Prozedurale Verfahren

Prozedurale Verfahren benötigen keine Ersterfassung von 3D-Daten, sondern erheben 3D-Gebäude im wahrsten Sinne des Wortes aus zweidimensionalen Grundrissdaten [vgl. Ledoux, Meijers 2009]. Für die Modellierung von Fassaden können attributierte Grammatiken („shape gramma") zum Einsatz kommen, welche leeren Flächen nach bestimmten Formeln mit vorgegebenen Objekten versehen [vgl. Müller et al. 2006; Krückhans, Schmittwilken 2009]. Die Massendaten eines Stadtmodells lassen sich dadurch deutlich schlanker prozessieren. Durch die Verwendung von künstlichen Texturen sehen die Gebäude jedoch nicht exakt so aus wie ihr entsprechendes Realobjekt [vgl. Müller et al. 2007; Döllner et al. 2005]. Ein prominentes Beispiel ist die Software *City Engine* von Procedural, die seit Sommer 2011 zum Portfolio von ESRI gehört [www9].

Einen etwas anderen Ansatz hat sich das Projekt *OSM-3D* auf die Fahne geschrieben [www10]. Da von der Community des OpenStreetMap-Projektes auch vermehrt einzelne Gebäude erfasst werden, liegt der Gedanke nahe, das Potential solcher so genannter „Volunteered Geographic Information" (VGI) [vgl. Goodchild 2007] auf die dritte Dimension zu projizieren. Objekte werden nur basierend auf ihren Attributen ("tags") modelliert z.B. die Extrusion des Grundriss um einen Höhenwert oder die Anzahl der Geschosse [vgl. Goetz, Zipf 2012a]. Derzeit wird in der OSM-Community viel über die Art und Weise, wie 3D-Informationen zu „mappen" sind, diskutiert, weil viele inhaltliche Anpassungen notwendig wären [www11]. Die Genauigkeit der bisherigen und kommenden Modelle wird gegenüber den anderen beschriebenen Methoden schwerer einzuschätzen sein. Doch OSM hat bereits in mehreren Studien seine Qualität im Vergleich zu proprietären Datensätzen bewiesen [vgl. Neis et al. 2010]. Warum sollte es in 3D nicht auch möglich sein?

2.3.3. Detailmodellierung

Ab einer Darstellung in LOD 3 kommen meist nur noch Einzelobjekte in einem Stadtmodell in Betracht, z.B. „Landmarks", die als repräsentative Gebäude das Stadtbild prägen [vgl. Döllner et al. 2006]. Im Bereich der Innenraummodellierung sind Building Information Models (BIM) mit dem Standard der Industry Foundation Classes (IFC) seit einigen Jahren in der Anwendung erprobt [vgl. Isikdag et al. 2012].

Für die detaillierte geometrische Erfassung von Gebäudekomplexen in LOD 3 und LOD 4 eigenen sich folgende Methoden:

- Die Aufnahme mit terrestrische Scannern [vgl. Pu, Vosselmann 2009]
- Die Modellierung aus CAAD-Datensätzen (Computer-aided architectural design) mit Programmen wie *3ds max* von Autodesk oder *SketchUp* von Google bzw. Trimble [vgl. Yin et al. 2009].
- Die Errechnung aus Fotos von speziellen 3D-Kameras [vgl. Lieckfeldt 2012]

2.4. Anwendungsgebiete für 3D-Stadtmodelle

In den letzten Jahren sind weltweit neue 3D-Stadtmodelle entstanden, die verstärkt auf offenen Standards aufbauen. Sie finden vermehrt Einsatz bei Anwendungen in Stadtplanung, Immobilienmärkten, Tourismus, Navigation, Umweltanalysen oder Simulation [vgl. Gerschwitz et al. 2011]. Zu den am häufigsten zitierten Beispielen im Zusammenhang mit CityGML gehören Lärmkartierungen [vgl. Czerwinski et al. 2007], Solarpotentialanalysen von Dächern [vgl. Ludwig, McKinley 2010] und Simulationen für ein Katastrophenmanagement [vgl. Randt et al. 2007].

2.4.1. Einbettung in einer GDI

Seit vielen Jahren werden weltweit Geobasisdaten aus analogen Plänen und Karten in die digitale Welt überführt und in etlichen Ländern in nationalen GDIs organisiert. In Deutschland verfügen die Landesvermessungsämter mit den digitalen Geodatenbeständen AFIS (Festpunkte), ALKIS (Liegenschaftskataster) und ATKIS (Topographiekataster) bereits über einen riesigen Pool aus georeferenzierten Grundrissdaten, der heute als AAA-Modell zusammengefasst wird [www12]. Die AdV hat entschieden, dass die dritte Dimension im AAA-Schema stärker eingebunden werden soll, und schreibt eine flächendeckende Haltung grundrisstreuer 3D-Modelle in LOD 1 ab dem 01.01.2013 für ganz Deutschland vor. Danach wird über einen Termin für die Haltung von LOD2-Modellen diskutiert [vgl. GeoInfoDok 2009]. Zwar verfügt dass AAA-Modell mit der Normenbasierten Austauschschnittstelle, kurz NAS, über ein einheitliches XML-Beschreibungsformat, für die Verwaltung von 3D-Stadtmodelle wird aber vorerst CityGML empfohlen [Gerschwitz et al. 2011: 292].

Auch in den Niederlanden wurde jüngst der Aufbau einer nationalen 3D-GDI konzipiert und geplant. Dabei wurde bisherige Technologien verglichen und für weitere Implementierungsschritte evaluiert, mit der Empfehlung CityGML zu verwenden [vgl. Stoter et al. 2011].

In der Europäischen Union wird der Aufbau von nationalen GDIs in den nächsten Jahren mit der *Infrastructure for Spatial Information in the European Community*, kurz *INSPIRE*, synchronisiert werden. Dieses Vorhaben beinhaltet ebenfalls dreidimensionale Geodaten. Die Datenspezifikation für Gebäude beruht zu großen Teilen auf den Kernelementen von CityGML [vgl. INSPIRE TWG BU 2011].

Kapitel 3

CityGML

Der Aufbau einer CityGML-Datei basiert auf der Auszeichnungssprache Extensible Markup Language (XML). Das trifft heutzutage für viele Formate zu, denn XML ist im Internetzeitalter DER quasi Standard zur Beschreibung verschiedenster Dateiformate geworden und dient als Informationsbus zwischen unterschiedlichen Systemen via Internet. Mit der Geography Markup Language (GML) wurde im Jahr 2000 von der OGC ein Standard festgesetzt, der XML um räumliche Datentypen, Filter und Operationen erweiterte, die wiederum auf Standards der ISO (191xx-Reihe) oder des OGC (z.B. Simple Features) beruhen [vgl. Herring 2011; Lake et al. 2000; Portele 2010, www13].

CityGML wendet GML an und erweitert seinerseits die Geometrieebene um eine thematische Ebene, um speziell Stadt- und Landschaftsmodelle nicht nur in ihrer Erscheinung sondern auch semantisch und topolgisch beschreiben zu können. Damit geht es weiter als andere auf Visualisierung ausgerichtete Formate für Stadtmodelle wie KML, COLLADA oder X3D [vgl. Caduff, Nebiker 2010]. Seit dem 13.08.2008 ist das CityGML-Schema als OGC-Standard angenommen [vgl. Gröger et al. 2008].

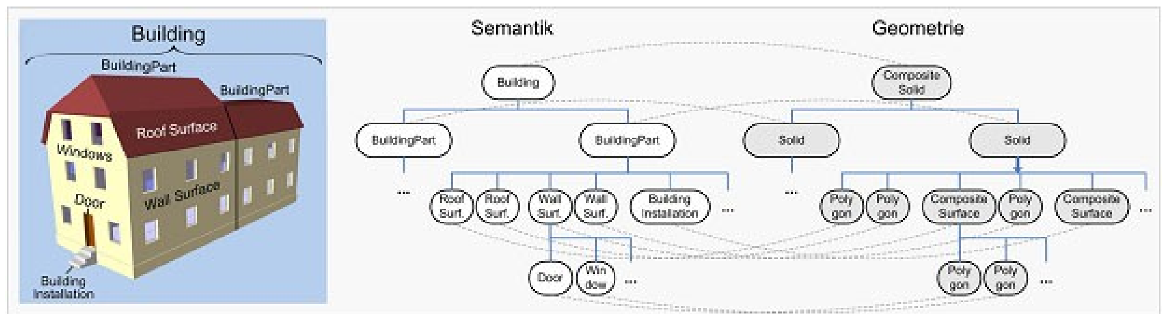


Abb. 3: Komplexes Objekt mit voll kohärenter räumlich-semantischer Struktur [Quelle: Stadler 2008]

Weitere prägnante Eigenschaften von CityGML werden in 3.1. und 3.2. bei der Beschreibung des Aufbaus verdeutlicht. 3.3. erklärt den Weg von der Spezifikation auf dem Papier zur Implementierung des Datenmodells und das Aufsetzen einer Datenbank. Anschließend wird die aktuelle Entwicklung des Standards in der Theorie (3.4) und die Einbindung in der Praxis (3.5) näher beleuchtet und gegen andere Format abgegrenzt (3.6).

3.1. Aufbau von CityGML

Die SIG 3D war bei der Entwicklung von CityGML bestrebt, möglichst viele Teilbereiche einer urbanen Infrastruktur zu berücksichtigen. Ein erster Blick auf Abbildung 4 lässt den breiten Funktionsumfang des Schemas erkennen.

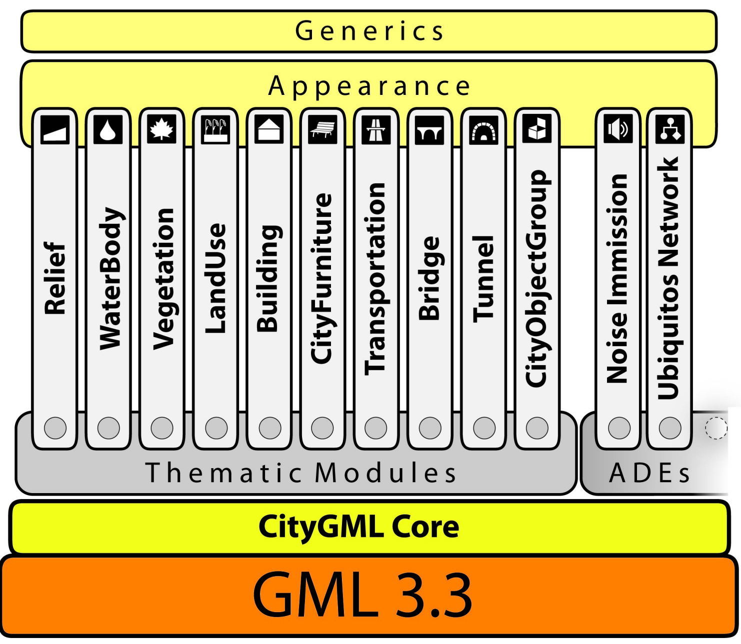


Abb. 4: Aufbau des CityGML-Anwendungsschemas

Die Säulen stellen optionale Erweiterungsmodule zu einem Datenmodellkern dar, d.h. für eine Implementierung von CityGML in eine Software sind neben dem Core-Modul nur noch das Appearance-Modul für die Erscheinung der Objekte und das Generics-Modul für benutzerdefinierte Attribute obligatorisch. Das Core-Modul enthält Identifikationsparameter, welche die thematischen Module erben. Über die Vergabe von externen Referenzen (z.B. Metadaten) können Objekte zwischen unterschiedlichen GDIs verlinkt werden [Kolbe 2009: 9]. Es können auch eigene Erweiterungsmodule über eine Schnittstelle, die „Application Domain Extension" (ADE), an das bestehende Schema eingefügt werden. Beispiele dafür finden sich in [Czerwinski et al. 2008, Hijazi et al. 2011, de Laat, van Berlo 2011]. Durch dieses modularisierte Konzept kann der Austausch zwischen unterschiedlichen System flexibler gestaltet und die Wiederverwendbarkeit der CityGML-Daten garantiert werden. Wie wichtig das sein kann, zeigt der mangelnde Erfolg des 3D-Grafikstandards Virtual Reality Modeling Language (VRML) [www14].

Durch die Modellierung in verschiedene LOD-Stufen (siehe 2.2.) ist CityGML hierarchisch strukturiert. In semantischer wie in geometrischer Hinsicht ergibt sich ein Vererbungsbaum, der mit jeder höheren LOD-Stufe um Klassen und Eigenschaften erweitert wird. Objekte können dementsprechend über mehrere Ebenen aggregiert werden. Umgekehrt sind die aggregierten Objekte durch ihre Bestandteile beschreibbar. Entweder rekursiv, d.h. ein Gebäudeteil eines Gebäudeteils eines Gebäudeteils, oder hierarchisch, d.h. ein Haus und seine Begrenzungsflächen, dessen Öffnungen zu bestimmten Räumen [Stadler 2008: 107]. Das erlaubt die redundanzfreie Integration von unterschiedlich detaillierten Datensätzen aus verschiedenen Quellen wie Stadtplanung, Facilitymanagement oder Forschung etc..

3.2. Das geometrische Modell von CityGML

Die geometrische Repräsentation von CityGML-Objekten basiert auf einem Oberflächenmodell, auch Boundary Representation (B-Rep) oder 2,5D-Repräsentation genannt. Entsprechend der Hierarchien im Datenmodell werden geometrische Körper (Solids) durch eine Aggregation ihrer Begrenzungsflächen beschrieben. Dies birgt den Vorteil, dass Löcher und Verschneidungen leichter zu verarbeiten sind, da nur die jeweils betroffenen 2D-Teilflächen betrachtet werden müssen und keine komplette 3D-Geometrie [vgl. Ellul, Haklay 2009]. Es handelt sich wieder um ein Konzept aus der Computergrafik [vgl. Foley et al. 1996], seit 2003 ist die B-Rep aber auch im Spatial Schema der ISO enthalten [vgl. ISO 2003a, Andrae 2008]. Durch das verwendete Oberflächenmodell muss CityGML nur eine Teilmenge der GML3-Geometrien abbilden, welche polygonale Geometrien in 2D und 3D umfasst. Dazu gehören Punkte, Linien, Flächen und Solids sowie die entsprechenden Composite- und Multi-Geometrietypen.

3.3. Vom Datenmodell zum relationalen Datenbankschema

Das Datenmodell von CityGML ist objekt-orientiert. Es wurde in vielen Treffen von Arbeitsgruppen der SIG 3D erarbeitet und mit der graphischen Modellierungssprache UML in Klassendiagramme umgesetzt. Objekt-orientierte Modelle folgen einem deutlich anderen Paradigma als relationale Datenbankschemata und können in der Regel nicht gänzlich verlustfrei gegeneinander abgebildet werden („impedance mismatch") [www15]. Für die Entwicklung der *3DCityDB* stand jedoch die Performanz im Vordergrund, weshalb beim Entwurf von Datenbank-Tabellen einige Abstraktionsschritte und Vereinfachungen der Objektklassen vorgenommen wurde, um die Anzahl der Entitäten zu verringern [vgl. Kolbe et al. 2009]. Folgende Maßnahmen waren dabei entscheidend:

- Datentypen der CityGML-Spezifikation mussten mit Datentypen des DMBS ausgetauscht werden. Komplexe Typen wie Codelisten, Vektoren und Matrizen wurden durch einfache Textvariablen („Strings") ersetzt.

- Attribute, für die beliebig viele Wertausprägungen definiert werden können, werden entweder mit einem Datentyp gesammelt beschrieben (meist in einem String mit Trennzeichen) oder in Felder mit vordefinierter Anzahl von Einträgen gespeichert.
- n:m-Beziehungen erfordern in einer relationalen Datenbank eine weitere Tabelle zwischen den Entitäten. Wo es möglich war, wurden Relationen zu einer 1:n-Beziehung umdefiniert, damit die Tabellenanzahl nicht zu stark anwuchs.
- Um die Kosten der Prozessierung bei rekursiven Relationen auf einem kalkulierbaren Level zu halten, wurden zusätzliche ID-Spalten hinzugefügt, die Verknüpfungen zu Elternelementen ausweisen (parent_id, root_id). Um z.B. alle zu einem Gebäude gehörigen Teile abzufragen, reicht die Angabe der parent_id des Gebäudes aus und vermeidet Joins über viele Tabellen.
- Anstelle einer Tabelle für die übergeordnete Basisklasse Feature lassen sich alle Objekte über GMLIDs, GML_NAME und NAME_CODESPACE genau identifizieren.
- Es war wichtig, dass Topologie- und Oberflächeneigenschaften separat abgefragt werden können, was für kombinierte Geometrien nicht möglich war. Mit der BRep-Aggregate-Klasse wurde eine Alternative modelliert, in der alle 2D- und 3D-GML-Geometrien mit Flag-Attrubuten wie isTriangulated, isSolid und isComposite als Teil einer Aggregation erkannt werden können und dennoch durch die Beschreibung als einzelnes Polygon identifizierbar bleiben [Nagel, Stadler 2008: 203]. Abbildung 5 verdeutlicht, wie das Geometriemodell durch die Konzeption erheblich vereinfacht wurde.

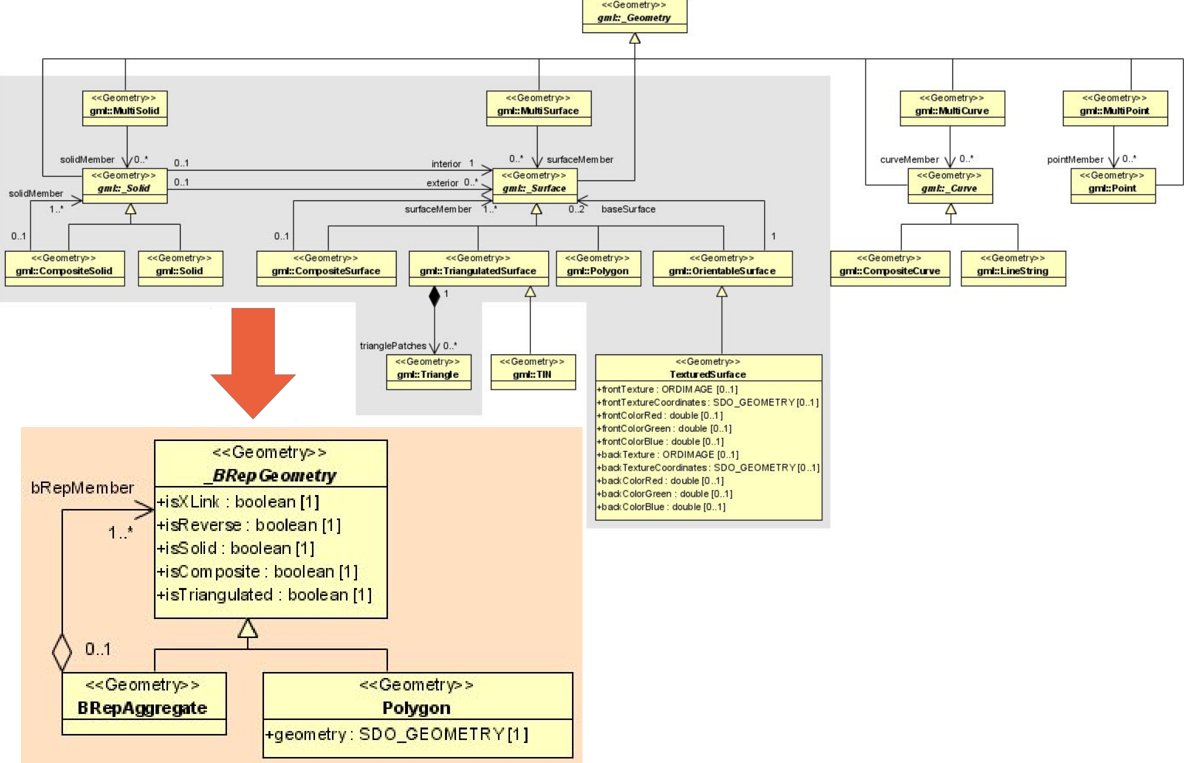


Abb. 5: Vereinfachung des Geometriemodells (aus *3DCityDB*-Dokumentation 2009: 18)

Nicht überall wurde wie bei der Geometrie die komplette Klassenhierarchie in eine Tabelle abgebildet. In einigen Modulen wurden mehrere zentrale Klassen entworfen z.B. CityObject im Core-Modul oder AbstractBuilding im Gebäudemodell. Abbildung 6 zeigt eine Gegenüberstellung von UML-Diagramm und Datenbankschema am Beispiel des Modells für das thematische Modul *Transportation*. Mithilfe des *JDevelopers* von Oracle wurden aus den Schemata die SQL-Skripte für die Tabellen generiert [Nagel, Stadler 2008: 200]. Damit war die Voraussetzung geschaffen die Inhalte eines CityGML-Instanzdokumentes in Datenbanktabellen aufzugliedern.

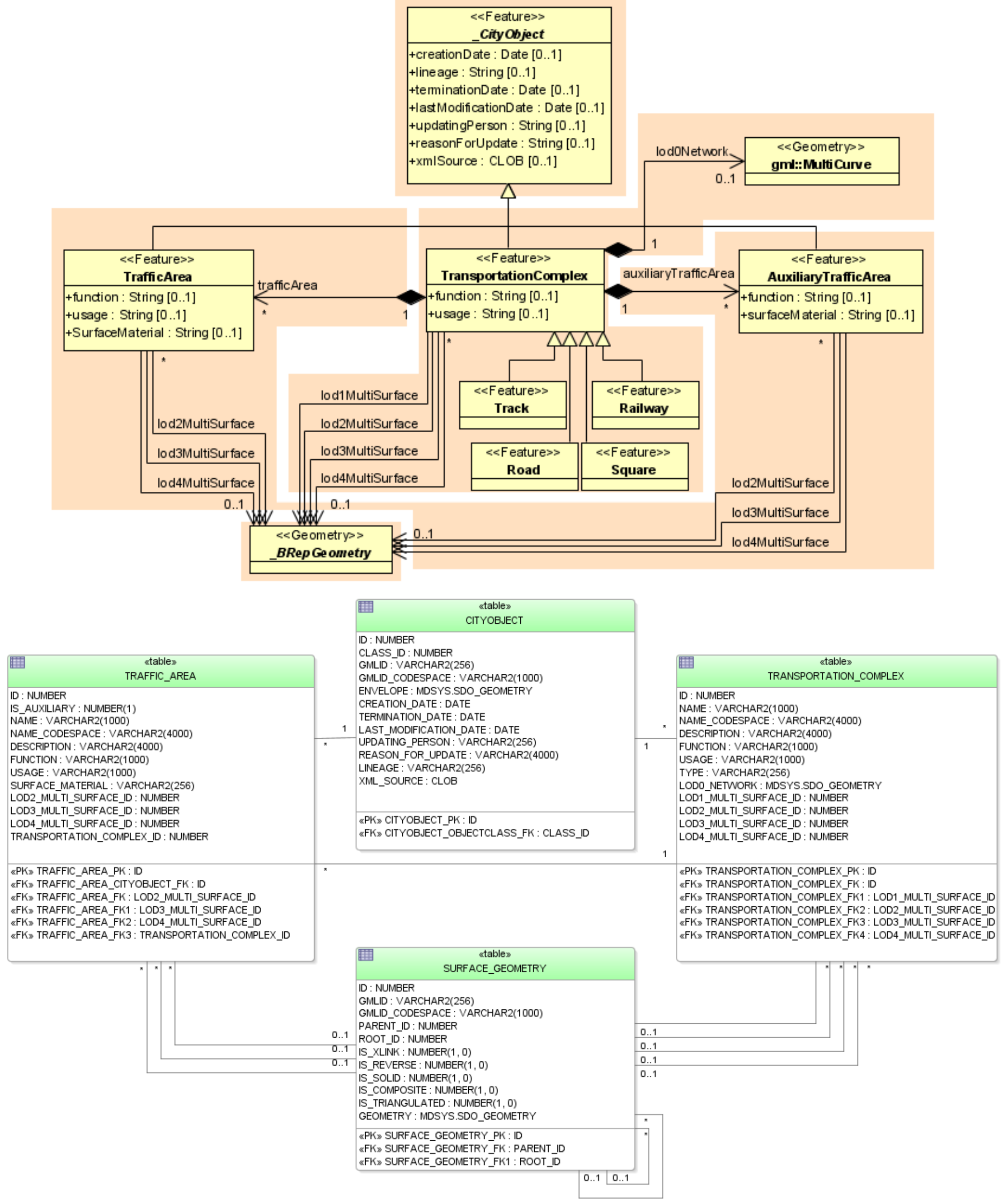


Abb. 6: Gegenüberstellung von UML-Modell und Datenbankschema am Beispiel des Transportation-Modells (aus *3DCityDB* Dokumentation 2009: 32, 64)

Es musste allerdings noch eine Anwendung für das DBMS entwickelt werden, das diesen Konvertierungsprozess in beide Richtungen ermöglicht. Die Entwickler entschieden sich für die Programmiersprache Java, weil mit *JAXB* (Java Architecture for XML Binding) und *SAX* (Simple API for XML) einfach zu implementierende aber dennoch mächtige Frameworks zur Verfügung stehen, um XML-artige Inhalte beliebiger Größe an Objekte der Programmiersprache zu binden, mit dessen Werten Datenbank-konforme Befehle geschrieben werden können [Nagel, Stadler 2008: 210]. Die entwickelten Methoden wurden in der Java-Klassen-Bibliothek *citygml4j* zusammengefasst, um eine Ausgangsbasis für alternative Java-Applikationen zu schaffen, denn Java hat sich in den vergangenen Jahren auch als eine führende Programmiersprache für GIS-Software etabliert [Zimmermann 2012: 281]. Die Verbindung zur Datenbank wurde über die *JDBC*-Schnittstelle (Java Database Connectivity) implementiert. Weitere technische Details werden in den Kapiteln 5.1. und 6.3. beschrieben.

3.4. Aktuelle Entwicklung

Standards sind keine starren Konstrukte. Sie müssen sich neuen Gegebenheiten anpassen und entsprechende Anforderungen erfüllen. Seit CityGML 1.0 verabschiedet wurde, sind etliche „Change Requests" von der OGC gesammelt worden. Sobald jedoch Veränderungen am Datenmodell nicht mehr mit früheren Versionen abbildbar sind, muss die Haupt-Versionsnummer erhöht werden. Lange wurde an einer abwärtskompatiblen Version 1.1.0 festgehalten. Da notwendige Änderungen am XML-Schema der OGC-Definition von Abwärtskompatibilität widerstrebten, wurde einer Version 2.0.0 der Vorzug gegeben. Am 24.04.2012 wurde CityGML 2.0.0 veröffentlicht [vgl. Gröger et al. 2012].

Die wichtigsten Änderungen zwischen den Versionen sind:

- Thematische Module für Brücken und unterirdische Strukturen (z.B. Tunnel)
- Zusätzliche Gebäudebegrenzungsfläche für eine semantische Klassifikation
- LOD 0-Representation für Gebäude (Grundriss, Dachkanten)
- Zusätzliche Attribute zum Beschreiben von Lagebeziehungen zu Land und Wasser
- Zusätzliche generische Attribute für Messwerte und Attribut-Sets
- Klassifizierende Attribute können ihre Werte nun aus beliebigen externen Codelisten erhalten und sind nicht mehr auf eine vordefinierte Liste beschränkt

3.5. Softwareunterstützung für CityGML

CityGML hat sich weltweit als Kontext- und Austauschformat für 3D-Stadtmodelle etabliert und wird daher von vielen entsprechende Anwendungen unterstützt. Ein Überblick findet sich im Wiki auf der CityGML-Homepage [www16] bzw. in Abbildung 7. Viele Produkte stammen wie der Standard selbst aus Deutschland. Die Spanne reicht dabei von den in Kapitel 2 beschriebenen 3D-Modellierungslösungen über XML-Validierer, 3D-Viewer in Desktop- oder Webumgebung bis hin zu GIS- und Datenbankschnittstellen wie der *3DCityDB*. Es existieren Konverter für verschiedene 3D-Formate zu CityGML, z.B.

IFC [vgl. El-Mekawy et al. 2011; *www17*], OSM3D [vgl. Goetz, Zipf 2012b], Shape3D und VRML [vgl. Kulawik et al. 2009]. *FME* von SafeSoft nimmt dabei eine herausragende Stellung ein. Es bildet in vielen Fällen die zentrale Schnittstelle für den Datenaustausch zwischen Geodatenbank und GIS (u.a. *ArcGIS*, *BentleyMap*). Unabhängig davon ist eine Stärke von *FME*, dass beliebig viele Datenquellen in selbst erstellten Konvertierungsmodellen („Workbenches") gebündelt und für ein CityGML-Dokument aufbereitet werden können. Dabei kann im Gegensatz zu den bisherigen automatisierten Erhebungsverfahren die gesamte inhaltliche Bandbreite des Datenmodells genutzt werden. Eine solche Modellierung ist jedoch alles andere als trivial oder gar intuitiv und setzt eine vertiefte Kenntnis des Datenmodells voraus. Benutzerfreundlichere Lösungen in Form einer Art Baukasten-Software speziell für CityGML existieren bisher nicht. *SketchUp* ist mittlerweile zu einer beliebten Ausweichmöglichkeit geworden, wenn es um das Editieren von einzelnen Gebäuden geht [www18, www19].

3DCityDB tridicon *CityGML-Toolchain* BIMserver **gvSIG** Aristoteles
LandXplorer *XML3D-CityGML-Converter* **FME** ***GO-Loader/Publisher*** CityGRID
IfcExplorer **BuildingReconstruction** *CityServer3D* FZKViewer 3DMap
BentleyMap *SketchUp-Import-Export-Plugin* novaFACTORY **QS-City 3D**
CodeSynthesisXSD **DB4GeO** SupportGIS/J-3D **CARD/1** *BS Contact Geo*
TerrainView INspector *Toposcopy* RCP **SpiderGIS** citygml4j ***libcitygml***

Abb. 7: Tag-Cloud mit Namen von Softwareprodukten, die CityGML verwenden

Viele in der Tag-Cloud auftauchenden Anwendungen bzw. Teile ihrer Produktfamilie bewerben 3D-Viewer. Die reine Visualisierung von virtuellen Städten und ihr exploratives Erkunden scheint eine unabdingbare wirtschaftliche Komponente geblieben zu sein. Anhand der Einstellung der *LandXplorer*-Produktserie von Autodesk [www20], dem bisher vorrangig genutzten Desktop-Viewer und Manager von Stadtmodellen im CityGML-Format, ist zu erahnen, dass die zukünftige Entwicklung in Web-Diensten weitergehen wird. Dafür werden schnell prozessierbare 3D-Formate benötigt, um für eine breites Publikum attraktiv zu sein (siehe 3.5.). Hierin hat sich eine Schwäche von CityGML offenbart. Die Instanzdokumente sind in ihrer Byte-Anzahl meist erheblich größer als Formate wie 3D-Shape, X3D/VRML oder KML (bis über 80%) [Kulawik et al. 2009: 294]. Selbst einzelne LOD3-Gebäude können schnell Gigabyte-Sphären erreichen.

Es gibt zwei Wege dem entgegenzuwirken. Entweder mit einer Generalisierung der Geometrien, wie sie u.a. von [Fan et al. 2009] vorgeschlagen wurde, oder das „Streamen" von Datensätzen. Dabei werden die Daten in ein gesondertes Binärformat gelesen, das nach erstmaligen Laden im Viewer Darstellungsinformation für die Grafikkarte des Rechners hinterlegt. Bei weiteren Aufrufen kann die Grafikkarte direkt auf die gespeicherten Vorgaben zugreifen und schont somit die Rechenleistung [vgl. Juen, Steffens 2012].

3.6. Vergleich zu anderen 3D-Formaten

Für ein schnelles Verstehen und Bedienen von Stadtmodellen bei Geschäftsprozessen werden CityGML-Dateien meistens ins Grafikformat KML überführt, da kein 3D-Viewer so stark verbreitet ist wie *Google Earth*. Deshalb wurde im letzten Jahr für den *3DCityDB-Importer/Exporter* eine Erweiterung für Exporte ins KML-/COLLADA-Format entwickelt [vgl. Kolbe et al. 2012].

Spannend bleiben Fortschritte von reinen Internetviewern auf Basis des X3DOM-Frameworks, das auf der 3D-Grafikbibliothek WebGL basiert und die Anbindung von dem Format X3D an HTML5 ermöglicht [vgl. Mao 2011; www21]. Ein simpleren Ansatz prägt das noch junge Projekt XML3D [vgl. Jochem 2012]. Standardisierungen für entsprechende Web-Services werden bei der OGC seit einigen Jahren mit dem Web-3D-Service (W3DS) und dem Web View Service (WVS) voran getrieben [vgl. Schilling, Kolbe 2010; Hagedorn 2010].

Tab. 1: 3D Austauschformate für Stadtmodelle im Vergleich
(vgl. Kolbe 2011: 59; Caduff, Nebiker 2010: 400; Stoter et al. 2011: 14)

Kategorie \ Standard	**X3D**	**COLLADA**	**KML**	**Shape**	**CityGML**	**IFC**	**DXF**
3D-Geometrie	+	+	o	+	+	++	+
Georeferenzierung	+	o	+	+	++	o	o
3D-Topologie	o	o	-	-	+	+	-
LOD	+	+	o	-	++	o	-
Texturierung	++	++	o	-	+	o	-
Semantik	o	o	o	+	++	++	o
Externe Links	+	++	++	-	++	-	-
Rendering	++	++	+	+	+	o	+
Einsatz im Web	+	+	++	-	+	-	-
XML-basierend	+	-	-	-	+	+	-
Akzeptanz	+	+	++	++	+	o	++
Erweiterbarkeit	+	++	+	-	++	o	-
offizieller int. Standard	W3C ISO	-	OGC	-	OGC	ISO	-

Art der Unterstützung: **++** sehr gut, **+** gut, **o** gering, **-** nicht vorhanden

CityGML kann und will kein Ersatz für die genannten Formate und Standards sein, sondern ist komplementär zu ihnen zu betrachten. Die viele Plussymbole in Tabelle 1 kennzeichnen nicht etwa die allgemein beste Lösung für alle Komponenten, sondern positionieren CityGML als zentrale Austauschschnittstelle zwischen verschiedenen speziellen Anwendungsfeldern von Stadtmodellen. Substitutionen ergeben sich meist für klassische Formate von proprietären Softwareanbietern. Viele Analysemethoden zu den in 2.4. erwähnten Themenfeldern können am umfassendsten mit CityGML selbst realisiert werden. Kapitel 4 wird erklären, warum dafür z.B. eine Geodatenbank in Frage kommt.

Kapitel 4

Wahl der Datenbank: Relationale DBMS

Rund 90% aller weltweit eingesetzten Datenbanken beruhen auf dem von [Codd 1970] entwickelten Konzept der relationale Entitäten. Selbst Paradigmenwechsel bei Programm- und Dateistrukturen hin zur Objektorientierung haben daran nichts geändert. Gerade bei komplexen Datentypen wie Geodaten stoßen Relationale DBMS (RDBMS) an ihre schematischen Grenzen. Objektorientierte DBMS (OODBMS) scheinen geeigneter zu sein [Bartelme 2005: 339], die Migration auf solche Systeme wird jedoch von vielen gescheut [Brinkhoff 2008:18]. OODBMS sind kaum verbreitet, Dokumentationen bzw. Onlinehilfen zwangsläufig begrenzt, personelle und finanzielle Aufwände folglich schwer kalkulierbar. Laut [Botha et al. 2011: 33] sind die Abfragen komplizierter und benötigen mehr Rechnerressourcen.

Aus dieser Situation heraus sind objektorientierte Erweiterungen für RDBMS entwickelt worden, z.B. die Kapselung von Objekten und die Vererbung bei Tabellen [Brinkhoff 2008: 20ff.]. Die Bezeichnung objekt-relationales DBMS (ORDBMS) wurde geprägt. Für eine Geodatenbank sind folgende Implementierungen notwendig:

- spezielle Geometrie-Datentypen
- Koordinatenreferenzsysteme
- eine Indizierung der Daten nach ihrer Lage
- geometrische und topologische Operationen

Seit 2005 ist der SQL-Zugriff auf Simple Features (OGC) spezifiziert (SQL/MM Spatial) [vgl. Herring 2010]. In vielen DBMS wurden Ansätze für eine Unterstützung von Geometrie-Datentypen geschaffen, aber nicht überall konsequent weiter entwickelt (siehe Kapitel 4.4.). *Oracle Spatial* und *PostGIS* können heute als die fortschrittlichsten räumlichen RDBMS angesehen werden [Zimmermann 2012: 249] und sind deshalb auch für die Umsetzung der *3DCityDB* in Frage gekommen [Kolbe et al. 2009: 13].

4.1. Oracle 11g mit Spatial Extension

Oracle ist mit seiner Datenbank seit Jahrzehnten marktführend. Die Ursprünge gehen auf die Forschungsarbeiten von Codd zurück. Die erste Version wurde 1979 veröffentlicht, die aktuelle 11g-Version (das „g" steht für Grid-Computing) im Jahre 2007 bzw. Release 2 in 2009. Die Produktvielfalt rund um die *Oracle Database* aufzuzählen wäre selbst als reine Stichwortsammlung an dieser Stelle zu umfangreich [www22]. Als Stärken werden oft die vielen Optimierungsmöglichkeiten (u.a. Result Cache, Materialized Views), die Verwaltung von sehr großen Datenmengen (Data Warehousing) und Schnittstellen für verteilte Systeme (Application Development) hervorgehoben [vgl. Ahrends et al. 2011a].

Die Anfänge einer Unterstützung für Geodaten fanden 1994 in der Version 7.1.6. mit der Speicherung von Punkten Einzug. In den folgenden Jahren kamen Linien, Polygone und räumlichen Operatoren hinzu und bildeten die *Spatial Cartridge* [Godfrind 2009: 3, Brinkhoff 2008: 31]. Ab Version 8i (1997) existiert die *Spatial Extension*, welche neue räumliche Datentypen (Spatial Data Object, kurz SDO), Koordiatensysteme (CRS) und lineare Referenzsysteme (LRS) einführte. Über die nächsten Jahre erweiterte man *Oracle Spatial* um Rasterunterstützung, Topologie- und Netzwerkanalysen, Kartenviewer und zuletzt 3D-Unterstützung mit Volumengeometrien, 3D-CRS und 3D-Indizes [vgl. Kothuri et al. 2007]. In nahezu allen Aspekten einer Geodatenbank war Oracle somit Vor- und Wegbereiter. Im Hauptteil der Arbeit werden einige Eigenschaften von *Oracle Spatial* genauer behandelt. Grenzen bei der Umsetzung nach *PostgreSQL* werden wiederum die besonderen Features der *Oracle Database* hervorheben u.a. das Workspace-Management.

4.2. PostgreSQL 9.1

PostgreSQL ist aus den Vorläufern *Ingres* (seit 1974) und *Postgres* (ab 1989) hervorgegangen und bezeichnet sich selbst gerne als die „fortschrittlichste Open-Source-Datenbank" [www23]. Im Marktanteil schlägt sich dieses Argument – auch gegenüber anderen Open-Source-Datenbanken – (noch) nicht nieder (siehe Abbildung 8).

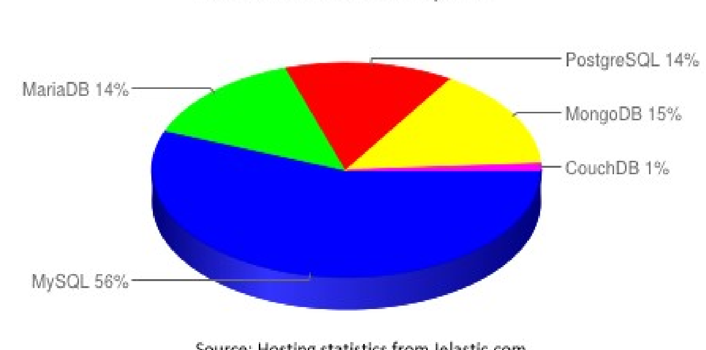


Abb. 8: Marktanteil von Open-Source-Datenbanken [www24]

Das hat natürlich wenig mit der Qualität des DBMS selbst zu tun. *PostgreSQL*-Anwender schwören auf die Stabilität und Schnelligkeit der Software, die professionelle Umsetzung verschiedener Verwaltungsfunktionen und die Verfügbarkeit von Schnittstellen zu mehreren Sprachen für server- und clientseitige Programmierung [vgl. Scherbaum 2009; Riggs, Krosing 2010]. Für Hilfestellungen kann auf eine umfassende Online-Dokumentation zurückgegriffen werden. Ansonsten können Fragen oder Problemberichte („Bug-Reports") auch über Foren, IRC-Chats oder Mailing-Listen verbreitet werden [www25], die dank einer aktiven Community meist zeitnah erörtert und gelöst werden.

Die Entwicklung wird durch ein quer über den Erdball verteiltes Entwicklerteam („Core-Team") voran getrieben, unterstützt von einigen Unternehmen wie Red Hat und EnterpriseDB. Die aktuelle Version ist seit September 2011 9.1, die wahrscheinlich im Herbst 2012 von 9.2 abgelöst werden wird. *PostgreSQL* hält sich sehr eng an den ISO SQL:2003 Standard [vgl. ISO 2003b] und erleichtert so die Migration von anderen RDBMS [Obe, Hsu 2011: 9]. Die Software selbst steht unter der Berkley-Software-Distribution-Lizenz (BSD), kann also verändert und ohne Quellcode-Freigabe vertrieben werden, muss aber auf das ursprüngliche Copyright verweisen [www26]. Ein Beispiel dafür ist *PostgreSQL Plus Advance Server* von EnterpriseDB [www27].

4.3. PostGIS 2.0

Seit dem Start im Jahre 2000 koordiniert Refractions Research die Entwicklung von *PostGIS*, die Geodatenbank-Erweiterung für *PostgreSQL* [www28]. *PostgreSQL* wurde als Fundament gewählt, weil die Erstellung von neuen Datentypen unterstützt wurde, eine spezifische Indizierung möglich war und darauf aufbauende Funktionen in Transaktionen konsistent verarbeitet werden konnten [Obe, Hsu 2011: 11]. Die strenge Einhaltung internationaler Standards wurde auch Vorbild für *PostGIS*, das sich eng an den Vorgaben von OGC (Simple Features) und ISO (SQL/MM Spatial) orientiert. Mittlerweile bietet *PostGIS* eine Bandbreite von über 900 räumlichen Operationen. Diese sind in der auf C basierenden *GEOS*-Bibliothek zusammengefasst, welche auch eine Portierung der *Java Topology Suite* (JTS) enthält [www29]. Weitere wichtige Bibliotheken sind *GDAL* für die Verarbeitung von Rasterdaten [www30] und *Proj4* [www31] für Koordinatensysteme. Die meisten Komponenten sind über Jahre in vielen Open-Source-GIS-Projekten zum Einsatz gekommen und daher auch vielen freien Entwicklern geläufig.

Mit der im April 2012 offiziell veröffentlichten Version 2.0 wird das Spektrum um viele Raster-, Topologie- und 3D-Funktionen erweitert. Dazu gehört auch die Integration des bisherigen Well-Known-Text-Raster-Projektes zum neuen Raster-Datentyp in *PostGIS* [vgl. Racine 2011] und die 3D-Geometrie-Typen Polyhedral Surface und TIN [vgl. Courtin 2011]. Außerdem unterstützt *PostGIS 2.0* das flexible Extension-System von *PostgreSQL*. Neue Datenbanken müssen nicht mehr mit einer *PostGIS*-Vorlage („Template") erstellt werden, sondern können die *PostGIS*-Erweiterungen nachträglich hinzufügen. Wie für *Oracle Spatial* folgen auch für *PostGIS* weitere technische Details im zweiten Teil der Masterarbeit.

An den abweichenden Veröffentlichungsterminen zur *PostgreSQL*-Version ist erkennbar, dass es sich bei *PostGIS* um ein eigenständiges Projekt handelt, hinter dem auch ein eigenständiger Entwicklerkern und eine eigene Community steht [www32]. *PostGIS* läuft unter der GNU General Public License (GNU GPLv3), d.h. Softwareableitungen müssen unter der selben Lizenz veröffentlicht werden und damit quelloffen sein [www33].

4.4. Andere RDBMS mit räumlichen Datentypen

Wie bereits erwähnt, gibt es neben *Oracle Spatial* und *PostGIS* noch weitere räumliche Add-Ons für RDBMS. IBM's *DB2* lässt sich durch den *Spatial Extender* erweitern, für das Tochterprodukt *Informix* gibt es das *Spatial DataBlade* und das *Geodetic DataBlade* [www34]. Trotz aller notwendigen Funktionen für eine Geodatenbank werden diese beiden Lösungen nur selten eingesetzt [www35].

Seit *SQL Server 2008* hat Microsoft eine *Spatial* Erweiterung nach OGC-Vorbild in alle Versionen integriert; auch in der kostenfreien Variante. Einführungen findet sich in [Atchinson 2009, 2012]. *SQL Server* verwendet einen andere räumliche Indexierung als *Oracle* und *PostGIS* (Multi-Level Grid) (für weitere Information siehe [Brinkhoff 2008: 192ff.]). Koordinaten-Transformationen werden leider nicht unterstützt.

MySQL, eine leistungsstarke, weit verbreitete Open-Source-Datenbank, ist zwar nur ein reines RDBMS, besitzt aber dennoch OGC-konforme räumliche Datentypen, Funktionen und Indizes [www36]. Der Funktionsumfang ist jedoch stark begrenzt und die Weiter-entwicklung stagniert. Außerdem können räumliche Operationen bislang nur auf Tabellen der MyISAM-Engine ausgeführt werden, die nicht in Transaktionen verarbeitet werden können. Das ist für ein produktiv arbeitendes System nicht praktikabel [Obe, Hsu 2011: 13]. Die Angaben sind äquivalent für die *MySQL*-Abspaltung *MariaDB* [www37].

SpatiaLite erweitert die gemeinfreie SQL-Programmbibliothek *SQLite* um geometrische Datentypen [www38]. Es verwendet die von *PostGIS* bekannten Bibliotheken *GEOS* und *Proj4* und hält sich an OGC-Standards. Eine Besonderheit gegenüber anderen Serverseitigen DBMS ist der dateibasierte Aufbau („Embedded"), der portable Datenbank-Lösungen auf mobilen Endgeräten ermöglicht. *SpatiaLite* kann in Verbindung mit *PostGIS* als eine Master/Slave-Anwendung genutzt werden, d.h. zur Synchronisierung von Feldarbeiten mit *SQLite* zu einem stationären *PostgreSQL*-Server [vgl. Kalberer 2010; Obe, Hsu 2011: 13].

Als letztes sei noch die *Spatial Database Engine* (SDE) von ESRI erwähnt. Sie stellt eine Middleware zwischen DBMS und *ArcGIS* dar. Aufgrund der starken Integration zu den am Markt weit verbreiteten *ArcGIS*- und *ArcServer*-Umgebungen findet die *ArcSDE* vielfach Anwendung. OGC- und SQL-Standards werden unterstützt und erleichtern die Verbindung zu Datenbanken wie *Oracle* und *PostGIS* [www39]. Im Gegensatz zur direkten Arbeit mit dem DBMS ergibt sich mit der *ArcSDE*-Middleware ein erhöhter administrativer Aufwand [www35].

4.5. NoSQL - Alternativen für die Speicherung von CityGML-Dateien?

Die Dominanz von RDBMS am Markt hat dazu geführt, dass Software zu alternativen Datenbankkonzepten heutzutage unter dem Begriff NoSQL („Not only SQL") zusammengefasst wird, auch OODBMS [vgl. Edlich et al. 2011]. Dokumentenorientierte bzw. XML-basierte Datenbanken sind dabei für die Speicherung von CityGML von Interesse, liegen doch die Instanzdokumente in genau jenem Format vor. Internetriesen wie Google, Facebook oder Amazon verwenden mittlerweile entsprechende nicht-relationale DBMS (meist Eigenentwicklungen). Die Hauptkompetenzen vieler NoSQL-Systeme sind die Verfügbarkeit zu jederzeit, d.h. alle Anfragen werden beantwortet, auch wenn Partitionen eines Datenbank-Netzwerkes ausfallen (Partitionstoleranz). Dafür kann jedoch nicht immer die Konsistenz der Daten nach dem ACID-Prinzip von RDBMS gewährleistet werden (Atomität, Konsistenz, Isolation, Dauerhaftigkeit). Dieses Szenario wurde von [Brewer 2000] mit dem CAP-Theorem erörtert.

Die Verwaltung von Geodatenbestände mit GIS-Werkzeuge hat sich in den letzten Jahren immer stärker ins Web verlagert. Durch die Verbreitung von XML-artigen räumliche Austauschformaten wie GeoJSON, SVG, GeoRSS oder das OSM-Format wurden auch Ansätze für Geo-Erweiterungen im NoSQL-Bereich entwickelt, oft in Form von räumlichen Indizes. *GeoCouch* ist ein eigenständiges und bisher einzigartiges Projekt, für die Entwicklung eines räumlichen Add-Ons für eine Dokumentendatenbank, in dem Fall *CouchDB* von Apache [www40]. Die Umfang an räumlichen Operationen ist noch minimal.

Native XML-Datenbanken wie *eXist* oder Oracle's *BerkleyDB* bieten den Vorteil eingehende XML-Daten unabhängig von ihrem Inhalt bzw. Anwendungsschema 1:1 in der Datenbank abzubilden. Die Datenbanken verbrauchen weniger Speicherplatz und Abfragen werden deutlich schneller verarbeitet als in einem RDBMS mit seiner Tabellen-struktur [Sondermann 2008: 108]. Die Inhalte sind mit den Abfragesprachen XPath oder XQuery leicht und effektiv erfassbar. Die Speicherung von GML-Dokumenten wird jedoch standardmäßig nicht unterstützt, weil GML nicht alle Teile des XML-Schemas unterstützt. Softwareerweiterungen sind demnach notwendig [Breunig, Zlatanova 2011: 792] und werden schon seit einigen Jahren besprochen.

In Zukunft könnte die noch junge Graphendatenbank *Neo4j* für CityGML interessant werden [www41]. Graphendatenbanken speichern nur Graphen und Knoten (‚in etwa vergleichbar zu Relationen und Entitäten eines RDBMS,) und beschleunigen die Arbeit vor allem bei rekursiven Datenmodellen wie sie z.B. bei CityGML vorliegen. *Neo4j* bietet bereits eine räumliche Indizierung und Schnittstellen zum *Geoserver* und dem Open-Source-GIS *uDig* (inkl. *GeoTools*-Bibliothek) an. Allerdings findet dies wahrscheinlich aufgrund wenig verbreiteter Anwenderkenntnisse von Graphendatenbanken bisher wenig Beachtung.

4.6. Verwandte Arbeiten

Wie in der Einleitung erwähnt, wurde in den Anfangsjahren der *3DCityDB* auch eine *PostGIS*-Version entwickelt. Bedingt durch eine Konzentration auf die technischen Anforderungen des Projektes um das Berliner Stadtmodell konnte diese jedoch nicht mehr berücksichtigt werden. [Fraysse 2009] nahm den Faden wieder auf und evaluierte die Umsetzbarkeit der weiterentwickelten *3DCityDB* auf *PostGIS*. Weitere Versuche wurden seitdem jedoch nicht unternommen.

Trotz eines allgemeinen Interesses an einer Speicherung von CityGML in *PostGIS* [Courtin 2011: 38] gab es bisher keine adäquaten Entwicklungen, die mit der *3DCityDB* vergleichbar wären. Allerdings existieren verwandte Ansätze für die Speicherung von GML-Modellen in Datenbanken. Die kommerziellen Lösungen *SupportGIS/J-3D* von CPA [www42] und *GO Loader* von Snowflake-Software [www43] können aus CityGML-Modellen generische Datenbankschemata für verschiedene RDBMS ableiten. Das entstandene relationale Schema ist jedoch nur kompatibel zu CityGML-Datensätzen mit gleichen oder geringeren Umfang an thematischen Objekten.

Mit dem Software-Paket *CityServer3D*, entwickelt vom Frauenhofer Institut in Darmstadt, werden heterogene Daten eines 3D-Stadtmodells (auch CityGML-Datensätze) in einem regelbasierten Datenmanagement harmonisiert und können u.a. in eine *PostGIS*-Datenbank gespeichert werden [www44].

Die Entwickler des interdisziplinären Projektes *MayaArch3D* verwenden in ihrem Werkzeug *QueryArch3D* ein an CityGML angelehntes Datenmodell zur Speicherung und Analyse von dreidimensionalen Modellen archäologischer Stätte. Als DBMS wird *PostgreSQL* mit *PostGIS* eingesetzt [vgl. Agugiaro et al. 2011].

Teil II

Die Portierung der 3DCityDB

Was erwartet den Leser?

In Kapitel 5 wird der bisherige Aufbau der *3DCityDB* vorgestellt, um die Auswahl der umzusetzenden Bestandteile besser nachvollziehen zu können.

Kapitel 6 beschreibt detailliert die notwendigen Anpassungen in drei Unterkapiteln:

- 6.1. SQL-Skripte des Datenbank-Schemas
- 6.2. Übersetzung der PL/SQL-Skripte
- 6.3. Java-Klassen des *Importer/Exporter*

Ausführliche Skript- und Code-Beispiele befinden sich im Anhang der Arbeit. Im Text werden die wichtigsten Spezifika beider ORDMBS nur exemplarisch mit wenigen Quellcode-Zeilen erklärt.

Kapitel 5

Anforderungsprofil für die PostGIS-Version der 3DCityDB

Bevor Bücher gewälzt, Artikel recherchiert und Zeilen programmiert wurden, bevor die Software installiert und sogar in neue Hardware investiert wurde, ging der Masterarbeit die Skizzierung einer Liste von notwendig Arbeitsschritten für eine Portierung voraus. Jeder Schritt bezog sich auf einen bestimmten Teil der *3DCityDB* in *Oracle* und verlangte ein äquivalentes Konzept in *PostgreSQL/PostGIS*. Nicht zuletzt dank der jüngsten Entwicklungen des *PostGIS*-Teams schien die Idee einer Umsetzung realisierbar zu sein.

5.1. Bisheriger Aufbau der 3DCityDB in Oracle

Die *3DCityDB* wurde für das 3D-Stadtmodell von Berlin entwickelt, welches im Rahmen des Projektes „Geodatenmanagement in der Berliner Verwaltung - Amtliches 3D-Stadtmodell für Berlin“ entstand und durch den Europäischen Fond für Regionale Entwicklung (EFRE) und dem Land Berlin finanziert wurde. Auftraggeber war die Berlin Partner GmbH und die Senatsverwaltung für Wirtschaft, Technologie und Frauen [vgl. Casper 2008; www45]. In der ersten Projektphase (Nov. 2003 – Dez. 2005) konzipierte das Institut für Kartographie und Geoinformation (IKG) an der Uni Bonn einen ersten Datenbank-Prototyp, der die bis dato grundlegenden Elemente von CityGML abbildete [vgl. Plümer et al. 2005]. Im vorangegangenen Kapitel wurde deutlich, warum für die Speicherung von CityGML ein RDBMS wie *Oracle Spatial* oder *PostGIS* am geeignetsten erschien.

In der zweiten Projektphase übernahm das Institut für Geodäsie und Geoinformationstechnik (IGG) an der TU Berlin die Weiterentwicklung und setzte 2007 mit Unterstützung der 3DGeo GmbH (heute Autodesk) die volle Abbildung des CityGML-Datenmodells auf ein Datenbank-Schema um. Zunächst auf Basis des OGC Best Practice Paper von CityGML 0.4.0, später dann auf der 1.0.0-Version [Kolbe et al. 2009: 9].

Das IGG entwickelte außerdem den *Importer/Exporter* und die dafür notwendigen PL/SQL-Skripte für datenbankseitige Prozeduren („Stored Procedures"). Um die Verbindungen der Anwendung mit der Datenbank nicht jedes Mal neu zu erstellen, zu initialisieren und zu schließen, werden sie in einem Verbindungs-Pool wie dem *Universal Connection Pool (UCP)* von Oracle vorgehalten, was zu spürbaren Performanz-Gewinnen beiträgt [Vohra 2008: 34]. Abbildung 9 abstrahiert den Aufbau der Datenbank.

Ausschlaggebend für die ausschließliche Unterstützung von *Oracle Spatial* war das Vorhandensein einer im Projekt geforderten Historienverwaltung in Form des *Oracle Workspace Manager*. Aufgrund der hohen Bedeutung dieses Features folgt ein kurzer Exkurs: Historienmanagement wird dann notwendig, wenn mehrere Benutzer denselben Datenbestand analysieren wollen und ihn dabei verändern müssen. Es treten zwangsläufig Konflikte auf. Diese werden vermieden, indem jedem Benutzer eine eigene Kopie der Daten zugewiesen wird, die nur er bearbeiten kann. Alle Tabellen müssen dafür in Sichten („Views") umgewandelt werden. Views können verändert werden, ohne dass die Originaldaten angerührt werden. Wenn die Änderungen in den Ausgangsdatenbestand, in *Oracle* der LIVE-Workspace, übernommen werden sollen, können sie mit ihm integriert werden (commit). Sollte ein anderer parallel arbeitender Benutzer danach dasselbe mit seinen Views vorhaben, müsste er entweder den Bestand auf den Zustand vor der letzten Änderung zurücksetzen („revert") oder die Konflikte zu dem Commit des anderen Bearbeiters prüfen und abwägen, welche Änderungen übernommen und welche verworfen werden. Der *Workspace Manager* ermöglicht auch eine Hierarchisierung von Datenkopien. Durch dieses System können verschiedene Versionen einer Datenbank aber auch verschiedene Datenstände über mehrere Jahre flexibel verwaltet werden. Es wird daher auch als Versionierung bezeichnet. Gerade im Bereich der Stadtplanung erfordern Analysen zu bestimmten Szenarien eine solche Historienverwaltung. In der *3DCityDB* wurde dies mit dem *Planning Manager*, einer Reihe von SQL-Skripten, realisiert.

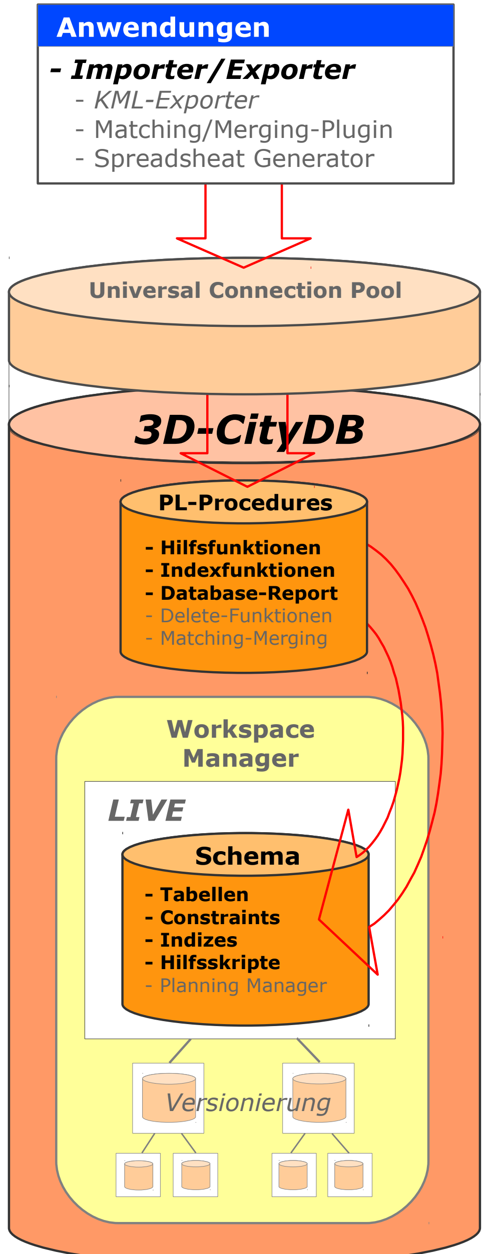


Abb. 9: Aufbau der 3DCityDB (**portierte Teile in schwarz**, nicht-portierte in grau)

Abbildung 10 bietet einen Überblick von der Ordner-Struktur mit den beinhaltenden Datenbank-Tabellen und wie sie ordnerübergreifend miteinander verknüpft sind.

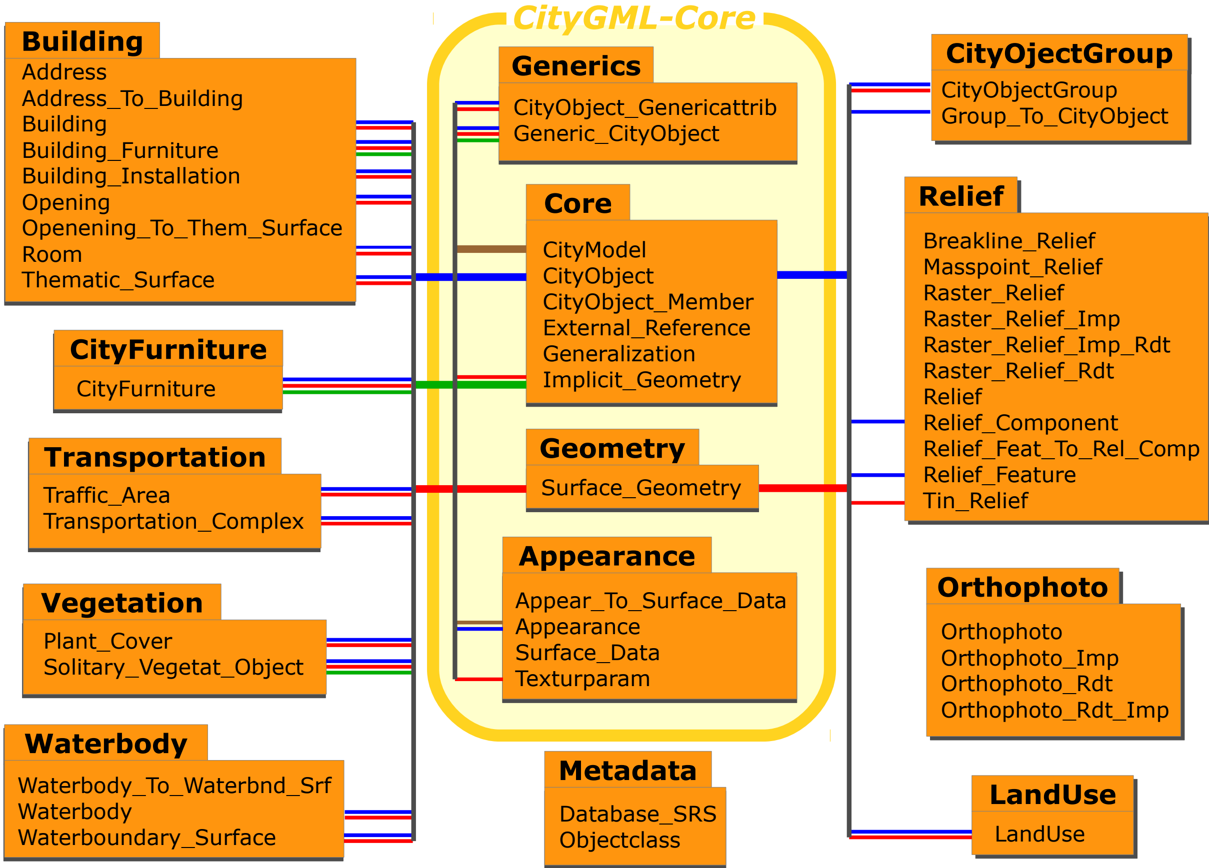


Abb. 10: Ordnerübergreifende Relationen zwischen Tabellen (Relationen zwischen Tabellen innerhalb eines Ordners sind nicht dargestellt, siehe dafür *3DCityDB*-Dokumentation.)

SQL-Skripte für die Erstellung von Sequenzen für ID-Spalten-, das Anlegen von Fremdschlüsseln („Foreign Keys“) und das Definieren von Indizes liegen in separaten Ordnern vor. Das gleiche gilt für die PL/SQL-Skripte, weitere Hilfsskripte („Utilities“) und den *Planning Manager*. Erstellt wird das Datenbank-Schema und die Prozeduren über das Aufrufen die SQL-Datei CREATE_DB (via *SQL*Plus* oder *SQL Developer*). Direkt nach dem Start muss das Referenzsystem (SRID) festgelegt, welches dann für alle Geometrie-Spalten gilt. Weitere Übersichten befinden sich im Anhang A.

5.2. Notwendige Umsetzungen im Rahmen der Masterarbeit

Es war zu bedenken, dass die *3DCityDB* und der *Importer/Exporter* über Jahre gewachsene Projekte sind, die fortlaufend in neuen Versionen erweitert und verbessert werden. Es stand daher von Beginn an fest, dass nicht alle Funktionalitäten im zeitlichen Rahmen der Masterarbeit umsetzbar sind, sondern das primäre Ziel die Speicherung und das Im- und Exportieren von CityGML-Dateien in eine *PostGIS*-Datenbank sein sollte.

In Abbildung 9 sind die dafür obligatorischen Komponenten schwarz hervorgehoben. Die zukünftige praktische Umsetzung der ausgeklammerten Bestandteile wird zum Abschluss der Arbeit in Kapitel 9 erörtert. Anhand des Aufbaus der *3DCityDB* sind gemäß den Zielvorgaben folgende Schritte für die *PostGIS*-Teilportierung durchzuführen:

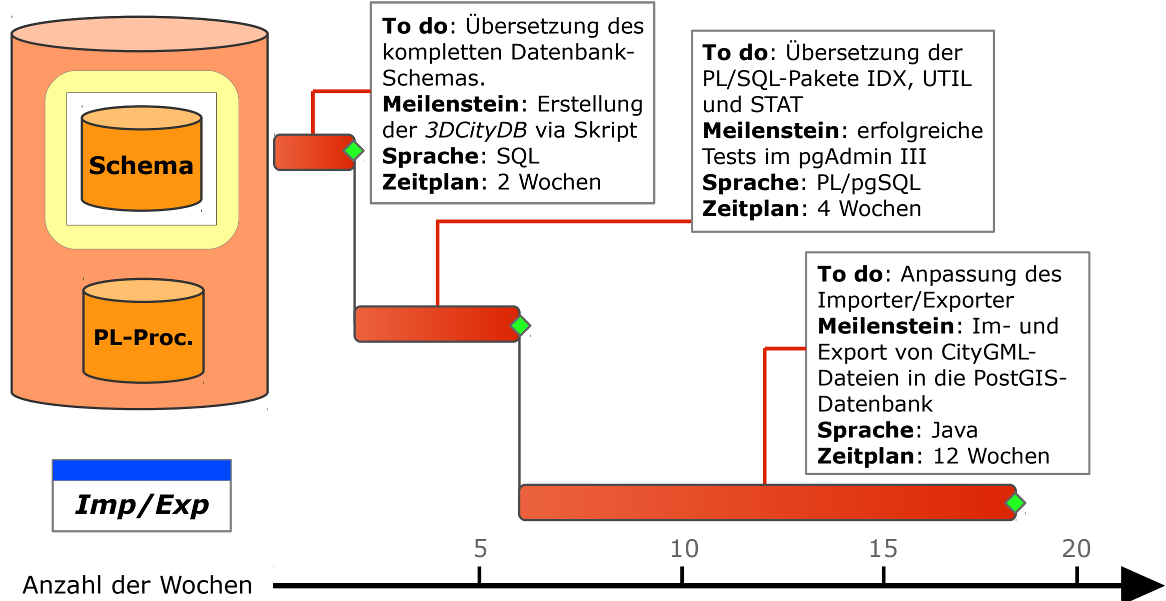


Abb. 11: Durchführungsplan der Portierung

Nach dieser Abfolge gliedert sich das nächste Kapitel.

Kapitel 6

Die Portierung auf PostgreSQL/PostGIS

Für die Portierung auf *PostGIS* konnte auf die Vorleistungen von [Fraysse 2009] zurückgegriffen werden. Diese beinhalteten eine nahezu komplette Übersetzung des Datenbankschemas und eine Teilumsetzung des *Planning Manager*. Die Skripte kamen allerdings nie zu einem dauerhaften Einsatz in der Praxis. Hier musste die Masterarbeit weitestgehend Pionierarbeit leisten.

6.1. Anpassungen der SQL-Skripte für das Datenbankschema

Die Migration von *Oracle Spatial* auf *PostGIS* ist, wie bereits erwähnt, vor allem aus finanzieller Hinsicht ein reizvoller Gedanke, und das nicht erst seit *PostGIS 2.0*. Da beide RDBMS größtenteils den selben Sprachstandard SQL implementieren, ist die reine Übersetzung eines Datenbankschemas sehr leicht, bei vielen Tabellen und Relationen jedoch zeitaufwendig. Häufig finden sich Anfragen nach speziellen Konvertern in Foren oder Mailing-Listen. Es gibt freie und kommerzielle Lösungen für *PostgreSQL* [www46, www47, www48]. Für *PostGIS* eignet sich momentan aber nur *FME*, siehe z.B. [Choquette 2010]. [Opsahl et al. 2012] haben als Alternative zu einer harten Migration eine Zwischenlösung entwickelt, die einen *Oracle*-Server in einer Master/Slave-Umgebung mit einem *PostgreSQL*-Server synchronisiert. Da bereits ein Großteil des Schemas übersetzt war, wurde auf eine automatisierte Variante verzichtet. Die letzten erforderlichen Anpassungen wurden manuell vorgenommen.

6.1.1. Unterschiede bei den Datentypen

Die verwendete Übersetzung bei verschiedenen Datentypen ist in Tabelle 2 aufgeführt. Die meisten Datentypen sind äquivalent zu einander. Eine gute Übersicht findet sich auch in [Singh 2011]. Funktionale Unterschiede werden auf den folgenden Seiten erläutert.

Tab. 2: Unterschiede bei den Datentypen

Oracle Spatial	**PostGIS** (PostgreSQL)	**Anmerkungen**
varchar2	varchar	
number	numeric	bei ID-Spalten integer
binary_double	double precision	
blob, clob	bytea, text	
	serial (integer)	in Oracle nicht vorhanden
ORDImage	bytea	ggf. auch PostGIS raster
sdo_geometry	geometry	
sdo_raster	raster	PostGIS raster entspricht stärker dem sdo_raster Typ
sdo_georaster		

Das Anlegen von Sequenzen

Für eine ID-Spalte wird in der Regel eine Sequenz erstellt, damit der Spaltenwert nicht bei jeden neuen Eintrag in die Tabelle angegeben werden muss. Die Sequenz vergibt nämlich automatisch eine ID, die inkrementell um 1 gegenüber dem Vorgängerwert erhöht wird. In *PostgreSQL* kann die Erstellung einer Sequenz implizit durch Zuweisung des Datentyps SERIAL im `CREATE TABLE` Befehl geschehen und muss nicht wie in *Oracle* explizit definiert werden. Da SERIAL von den Datentyp INTEGER erbt, müssen alle Spalten, welche sich auf die entsprechende ID-Spalte referenzieren, vom Typ INTEGER sein. Bei NUMERIC würde eine Fremdschlüssel-Verbindung fehlschlagen.

Das Speichern von Bildern

Ein großer Unterschied ergibt sich bei dem *Oracle*-spezifischen Datentyp ORDImage, der zum Speichern von Bildern dient. ORDImage ist Teil der *Oracle Multimedia* Komponente und bietet Funktionen ähnlich dem eines Grafikbearbeitungsprogramms an. In der *3DCityDB* wird es für die Speicherung von Texturen eingesetzt. In *PostgreSQL* gibt es kein vergleichbares Konzept. Mit dem Datentyp BYTEA, der *PostgreSQL*-Version eines Binary Large Object (BLOB), ist nur die für relationale Datenbanken herkömmliche binäre Speicherung von Bildern möglich, die keine inhaltlichen Abfragen erlaubt [vgl. Haberl, Müllenbach 2009]. Da die *3DCityDB* von den Möglichkeiten des ORDImage-Datentyps kaum Gebrauch macht und BLOBs weniger Speicher belegen, bestehen seitens der *3DCityDB*-Entwickler Überlegungen auch auf *Oracle*-Seite auf die BLOB-Speicherung umzusteigen.

Geometrien in der Datenbank

Bei der Speicherung der Geometrien geht Oracle ebenfalls eigene Wege. Wie in *PostGIS* existiert ein zu SQL/MM Spatial konformer Datentyp ST_GEOMETRY (**S**patial **T**emporal), geläufig ist jedoch die Verwendung von SDO_GEOMETRY (**S**patial **D**ata **O**bject), einem aus fünf Attributen zusammengesetzten Datentyp. Dieser beinhaltet Metadaten zu Dimension (erste Ziffer vom `GTYPE`), Referenzsystemen (`SRID`) und Art der Geometrie

(letzte Ziffer vom GTYPE, ETYPE und INTERPRETATION für Sonderformen). Detaillierte Erklärungen finden sich in der Online-Dokumentation [www49] oder in [Brinkhoff 2008: 99ff.]. Ein Beispiel: Für ein Quadrat mit den Koordinaten (0,0), (0,20), (20,20), (20,0) kann der Tabellen-Eintrag wie folgt aussehen:

Tabelleneintrag:	**Bestandteile:**
`SDO_GEOMETRY(`	`SDO_GEOMETRY(`
`2003,3068,NULL,`	`GTYPE, SRID, POINT,`
`SDO_ELEM_INFO(1,1003,3),`	`SDO_ELEM_INFO(STARTING_OFFSET,ETYPE,INTERPRETATION)`
`SDO_ORDINATES(0,0,20,20))`	`SDO_ORDINATES(ORDINATE_ARRAY))`

Bei *PostGIS* findet sich zum selben Quadrat folgender Eintrag in der Geometriespalte:

```
01030000A0FC0B000001000000050000000000000000000000000000000000000000000000000000000000000000000000
00000000000034400000000000000000000000000000344000000000000034400000000000000000000000000000344000
000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Das entspricht dem Well Known Binary (WKB) Format.

Tab. 3: Unterschiede bei den Geometrie-Datentypen

Attribut	**Oracle SDO_GEOMETRY**	**PostGIS GEOMETRY**
Typisierung	`GTYPE` typisiert Dimension, Geometrieart, bildet Vorschrift für Aufbau: Polygon = 2003, 3D-Polygon = 3003	nach OGC-Simple-Features wird die Geometrie mit Worten bezeichnet: `POLYGON((...)), POLYGONZ((...))`
SRID	ist ein eigener Bestandteil	ist Teil des erweiterten WKT-Strings
Form und Regeln der Geometrien	`SDO_ELEM_INFO` gibt Auskünfte über die Form der Geometrie (auch Aggregate) und definiert Regeln für die Anordnung der Ordinaten. Bei äußeren Ringen von Polygonen (`ETYPE` = 1003) müssen Punkte im Uhrzeigersinn angeordnet sein, bei Innenringen (`ETYPE` = 2003) umgekehrt. Wann der Innenring im Ordinaten-Array beginnt ist mit `STARTING_OFFSET` beschrieben: `ELEM_INFO(1,1003,1,`**`11`**`,2003,1)`	es gibt keine genauere Beschreibung als die Typisierung, welche wie bei *Oracle* den Aufbau vorschreibt. Die Orientierung der Ringe funktioniert jeweils in beide Richtungen. Innenringe werden daran erkannt, dass sie in weiteren Klammerblöcken folgen: `POLYGON((`Außenring`)(`Innenring`)(...))`
Sonderfall Rechtecke (optimized rectangles)	Rechtecke können wie im obigen Beispiel auf zwei Punkte reduziert werden, wenn die `INTERPRETATION` das Objekt als Rechteck auszeichnet: `(1,1003,`**`3`**`)` Langfristig sollten Rechtecke aber als Polygone gespeichert und in optimierter Form nur zu Filterzwecken eingesetzt werden.	alle Punkte müssen genannt oder die Datentypen Box, Box2D oder Box3D verwendet werden: `BOX(p1,p2)`
Struktur der Koordinaten	alle Ordinaten der Geometrie sind durch Kommata getrennt und werden (auch bei Aggregaten) in einem Array gelistet: `(x1, y1, x2, y2, ... )`	im WKT-Format wird nur zwischen Punkten ein Komma gesetzt. Die Ordinaten sind mit Leerzeichen getrennt: `(x1 y1,x2 y2, ... )`

Beim Anlegen von Geometriespalten musste in *PostGIS* stets auf die Funktion `AddGeometryColumn` ausgewichen werden. Seit der Version 2.0 können diese wie bei *Oracle* auch in einer `CREATE TABLE`-Definition stehen. Metadaten wie Geometrietyp, Dimension und Referenzsystem folgen als „type modifier" in den Klammern. In beiden Varianten erfolgt implizit ein Eintrag im View `geometry_columns`, dem Pendant zur Tabelle `USER_SDO_GEOM_METADATA` in *Oracle*. Bei *Oracle* muss der Eintrag jedoch manuell vorgenommen werden. Ein kurzer Vergleich:

```
CREATE TABLE surface_geometry(
  id          NUMBER NOT NULL,
  geometry    SDO_GEOMETRY,
  . . .
)

INSERT INTO USER_SDO_GEOM_METADATA (TABLE_NAME, COLUMN_NAME, DIMINFO, SRID)
  VALUES ('SURFACE_GEOMETRY', 'GEOMETRY',
    MDSYS.SDO_DIM_ARRAY
      (MDSYS.SDO_DIM_ELEMENT('X', 0.000, 10000000.000, 0.0005),
       MDSYS.SDO_DIM_ELEMENT('Y', 0.000, 10000000.000, 0.0005),
       MDSYS.SDO_DIM_ELEMENT('Z', -1000, 10000, 0.0005)) , 3068);
```

ORACLE DATABASE 11g

```
CREATE TABLE surface_geometry(
  id          SERIAL NOT NULL,
  geometry    GEOMETRY(PolygonZ,3068),
  . . .
)
```

PostgreSQL PostGIS

oder

```
CREATE TABLE surface_geometry(
  id          SERIAL NOT NULL,
  . . .
)

SELECT AddGeometryColumn('surface_geometry', 'geometry', 3068, 'POLYGON', 3);
```

Mit der Funktion `ST_AsEWKT(geometry)` wird die Geometrie im leserlichen Extended Well Known Text (EWKT) angezeigt, hier `SRID=3068;POLYGON((0 0,0 20,20 20,20 0,0 0))`. WKT entspricht der Schreibweise der Simple Features und ist auch in *Oracle* für SDO_GEOMETRY-Einträge darstellbar. Für `INSERT`-Befehle muss auf die Methode `ST_GeomFromEWKT('siehe EWKT-Beispiel')` oder `ST_GeomFromText(POLYGON((..)),3068)` zurückgegriffen werden. In *Oracle* können die Attribute von SDO_GEOMETRY hingegen direkt nach derselben Struktur wie in den Zeileneinträgen definiert werden. Die Handhabung ist zwar insgesamt komplexer als bei *PostGIS*, erlaubt aber eine differenziertere Beschreibung der Geometrien, was besonders in Java nützlich wird.

Die Verwaltung von Rasterdaten

In *Oracle Spatial* werden n-dimensionale Raster mit dem Datentyp SDO_GEORASTER beschrieben, der sich aus einer Matrix von Zellen mit ID, Rastertyp, der geometrischen Ausdehnung und Metadaten in Form einer XML-Datei zusammensetzt. Jedes SDO_GEORASTER-Objekt ist mit mindestens einem SDO_RASTER-Objekt verknüpft. Für alle zu einem GEORASTER gehörigen SDO_RASTER-Objekte muss eine Tabelle angelegt werde, welche die Struktur und Werte der Rasterdaten speichert. Diese „Raster Data Table" (RDT) enthält die Blockstrukturen von GEORASTER-Objekten in 2D (z.B. Kacheln), Pyramidenebenen und die eigentlichen Bilddaten als BLOBs. Zum Einladen von Rasterdaten können verschiedene, meist über die Kommandozeile aufzurufende Anwendungen benutzt werden [vgl. Brinkhoff 2008: 403ff; Kothuri et al. 2007:725ff; www50].

Die Verwaltung von Rasterdaten in *PostGIS* ist ähnlich aufgebaut wie bei den Vektor-Geometrien und weit weniger komplex als in *Oracle*. Es gibt den Datentyp RASTER und eine Verknüpfung zum View `raster_columns`, in dem die Metadaten der Rasterdaten aufgeführt sind. Bisher lassen sich nur 2D-Raster speichern. Kacheln werden als eigenständige Rasterdateien behandelt und nicht als Matrix für ein RASTER-Objekt. Für Pyramidenebenen müssen Rasterdaten in ein separates Objekt mit geringerer Auflösung gespeichert werden. Diese Ebenen werden dann im View `raster_overviews` verwaltet. In die Datenbank gelangen Rasterdaten mit Hilfe der C-Funktion *raster2pgsql*, die über die Kommandozeile ausgeführt werden muss. Sie erzeugt gemäß den gesetzten Parametern eine SQL-Datei mit passendem `INSERT`-Kommando [www51]. Ein Import in die Tabelle `raster_relief` könnte z.B. mit diesem Befehl erfolgen:

```
raster2pgsql -f rasterproperty -s 3068 -I -C -F -t 128x128 -l 2,4 relief/*.tif
     raster_relief > rastrelief.sql
```

Alle Rasterdaten aus einem beliebigen Ordner relief/*.tif werden in der Tabelle `raster_relief` in die Spalte `rasterproperty` geladen (`-f`), nach dem Soldner-CRS referenziert (`-s 3068`) und mit einem räumlichen Index indiziert (`-I`). Jedes Raster wird gekachelt (`-t`), erhält Constraints (`-C`) und Pyramidenebenen (`-l`, Level 2 und 4). Die Tabelle wird um eine weitere Spalte mit dem Dateinamen datei.tif (`-F`) erweitert. In der generierten SQL-Datei können dann noch nachträglich nicht vergebene Werte für die restlichen Attribute der Tabelle hinzugefügt werden, bevor sie ausgeführt wird. Ist ein SRID vergeben worden, wird in dem `raster_columns`-View der geometrische Rahmen („Extent") errechnet.

Für die Portierung konnten neben den *Oracle*-spezifischen RDT-Tabellen auch alle Tabellen mit der Endung IMP und IMP_RDT entfallen, welche einst als Importlogik für einen von der Uni Bonn entwickelten Raster-Importer angelegt worden waren. Insgesamt konnten sechs Tabellen aus den Ordnern Relief und Orthophoto eingespart werden.

6.1.2. Constraints und Trigger

Constraints sind Bedingungen, die definiert werden, um die Integrität und Konsistenz der Daten innerhalb einer Datenbank zu wahren. Die Deklarationen ist nahezu identisch zwischen den Systemen. Wird eine Bedingung bei dem Versuch den Inhalt einer Tabelle zu verändern verletzt, ruft dieses Ereignis einen impliziten Trigger auf, der dies verhindert. Diese Trigger können in beiden DBMS mit leicht unterschiedlichen Methoden deaktiviert werden. Es können auch eigene Trigger definiert werden. In der *3DCityDB* war dies für die Verwaltung der RDT-Tabellen notwendig, die jedoch, wie eben beschrieben, in der *PostGIS*-Version entfallen.

6.1.3. Unterschiede bei räumlichen Indizes

Die Werte aus Spalten mit sortierbaren Datentyp-Literalen wie IDs oder Namen werden in *Oracle* und *PostgreSQL* jeweils mit einem B-Baum-Index strukturiert. Die Definition eines B-Baum-Indexes ist identisch:

```
CREATE INDEX cityobject_fxk ON cityobject (class_id);
```

Geometrien lassen sich jedoch nicht auf diese Weise organisieren und benötigen einen Index, der die Daten nach ihrer Lage hierarchisiert. Hierfür gibt es verschiedene Ansätze. Bei einem R-Baum-Index reicht als Identifikation einer Geometrie das Abspeichern ihres kleinsten umschließenden Rechteckes („Bounding Box"), siehe Abbildung 12.

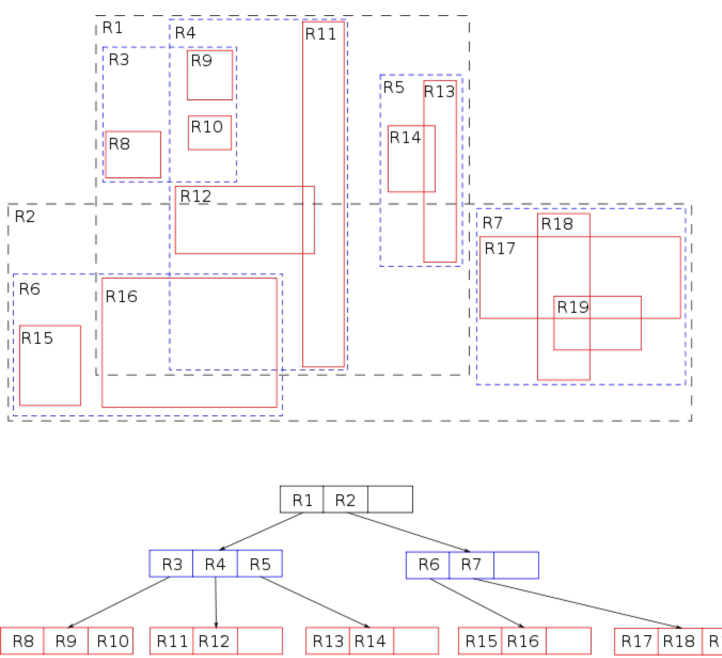


Abb.12: Indizierung von Geometrien mittels ihrer Bounding Boxes [Quelle: wikipedia]

In *Oracle Spatial* wird der R-Baum-Index nach [Guttman 1984] verwendet. *PostgreSQL* implementiert mit dem Generalised Search Tree (GiST) nach [Hellerstein et al. 1995] eine allgemeinere Form zur Speicherung von irregulär strukturierten Daten. Seine interne Struktur kann so konfiguriert werden, dass er sich wie ein R-Baum-Index verhält, was für Geometrien in *PostGIS* benötigt wird [vgl. Breunig, Zlatanova 2011: 793ff; Brinkhoff 2008: 161ff; Yeung, Hall 2007: 115]. Die Definitionen unterscheiden sich geringfügig.

```
CREATE INDEX cityobject_sxk ON cityobject(envelope) INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

```
CREATE INDEX cityobject_sxk ON cityobject USING GIST (envelope gist_geometry_ops_nd);
```

Die Dimension des Indexes ergibt sich bei *Oracle* aus den Metadaten der Geometriespalte (siehe 6.1.1.). Bei *PostGIS* spezifiziert erst die Angabe `gist_geometry_ops_nd` die n-dimensionale Indexierung von Geometrien. In der *3DCityDB* sind 3D-Indizes wichtig für die korrekte Ausführung von dreidimensionalen räumlichen Operationen (z.B. Distanz-messungen, Verschneidungen, topologische Untersuchungen etc.). Damit die Indizes auch garantiert verwendet werden, wird empfohlen nach einem Datenimport explizit Tabellen-Statistiken für die Geometrie-Spalten von *PostgreSQL* erstellen zu lassen. Dies geschieht mit dem Befehl `VACUUM ANALYSE`. Um den Benutzer die manuelle Eingabe für alle betroffenen Tabellen zu ersparen, wurde eine entsprechende SQL-Datei geschrieben.

6.1.4. CREATE_DB.sql

Die Erstellung des Datenbankschemas kann in *PostgreSQL* nur über *psql* erfolgen. Im *pgAdminIII* ist das Ausführen von Dateien aus einem Skript heraus nicht möglich. Die Arbeit auf der Kommandozeile sollte für den Benutzer jedoch so gering wie möglich gehalten werden. Deshalb wurde eine einfache Batch-Datei programmiert, die das CREATE_DB.sql-Skript ausführt. In dem Skript werden mit dem Operator `\i` (bei *Oracle* @) alle weiteren SQL-Dateien aufgerufen werden und die Erstellung des *3DCityDB*-Schemas vervollständigt. In der Batch-Datei müssen nur die Verbindungsparameter für eine zuvor angelegte leere *PostGIS*-Datenbank editiert werden. Das SRID und der GML-SRS-NAME werden wie bei der *Oracle*-Variante über Benutzereingaben gesetzt und in Variablen an die anderen Skripte weitergereicht. Default-Werte können leider nicht definiert werden. Das Löschen des Schemas wird auch über eine Batch-Datei gesteuert.

6.2. Übersetzung der PL/SQL-Skripte

PL/SQL ist eine proprietäre Sprache von Oracle und stellt eine prozedurale Erweiterung von SQL dar. Sie ermöglicht Variablen und Kontrollstrukturen (Bedingungen, Schleifen etc.), wodurch Abfrageprozesse automatisiert werden können [vgl. Skulschus, Weiderstein 2011]. In der *3DCityDB* wurden PL/SQL-Prozeduren erstellt, um die Verbindungsanzahl durch den *Importer/Exporter* mit einer Bündelung von Prozessen auf Datenbankseite zu minimieren.

Die zu portierende Teile gliedern sich in folgenden Pakete:

- IDX: zum Erstellen, Entfernen und Abfragen des Status von Indizes
- UTIL: Hilfsfunktionen für verschiedene Operationen
- STAT: zählt die Einträge der Tabellen für einen Report im *Importer/Exporter*

Das Konzept der prozeduralen Erweiterung von SQL wird heute von vielen Datenbankherstellern implementiert und orientiert sich am Persistent Stored Module des SQL Standards (SQL/PSM) [vgl. ISO 2003b]. In *PostgreSQL* ist es die Sprache PL/pgSQL, die nicht nur im Namen eine sehr hohe Ähnlichkeit zu PL/SQL aufweist. Die Unterschiede müssen an dieser Stelle nicht genauer beschrieben werden, weil das folgende exemplarische Code-Beispiele diese ausreichend gut demonstriert.

```
CREATE FUNCTION example_fct (param NUMBER := 0) RETURN VARCHAR2
IS
  var1  NUMBER;
  var2  VARCHAR2 := 0;
  var3  VARCHAR2 := 0;
BEGIN
  IF param <> 0 THEN
    EXECUTE IMMEDIATE 'SELECT num_val, char_val FROM table WHERE id=:1'
      INTO var1,var2 USING param;
  END IF;

  procedure_executed_here;

  var3 := example_fct2(var1,var2);

  EXCEPTION
    WHEN OTHERS THEN
      dbms_output.put_line('An error occured: ' || SQLERRM);

  RETURN var3;
END example_fct;
/
```

```
CREATE FUNCTION example_fct (param NUMERIC DEFAULT 0) RETURNS VARCHAR AS $$
DECLARE
  var1  NUMERIC;
  var2  VARCHAR := 0;
  var3  VARCHAR := 0;
BEGIN
  IF param <> 0 THEN
    EXECUTE 'SELECT num_val, char_val FROM table WHERE id=$1'
      INTO var1,var2 USING param;
  END IF;
  PERFORM procedure_executed_here();

  var3 := example_fct2(var1,var2);

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'An error occured: %', SQLERRM;

  RETURN var3;
END;
$$
LANGUAGE plpgsql;
```

Es liegt nahe, dass auch für PL/SQL Konverter existieren, um die Migration nach *PostgreSQL* zu beschleunigen [vgl. Darold 2011; Rosensteel 2011]. Die Bewertung der Ergebnisse einer Konvertierung setzt jedoch ein Grundverständnis für beide Sprachen voraus. Der Lernprozess war nach dem Empfinden des Autors besser durch die manuelle Übersetzung voran zu treiben, zumal sich die Anzahl an Funktionen noch in einem überschaubaren Rahmen hielt. Die Portierung der reinen Sprachblöcke ist, wie am Beispiel zu sehen, eine leichte Angelegenheit, wenn die äquivalenten Schlüsselwörter bekannt sind. Zwei Aspekte stellten jedoch eine größere Herausforderung dar und werden in den folgenden Unterpunkten kurz erläutert.

6.2.1. PL/SQL Packages versus PostgreSQL Schemata

Beim Programmieren von Stored Procedures ist es häufig nicht zu vermeiden, dass bestimmte Funktionsblöcke in Hilfsfunktionen ausgelagert werden. In der Summe können dadurch viele Funktionen entstehen, die in einer für alle berechtigten Benutzer öffentlichen Schnittstelle für sich selbst genommen keinen Zweck erfüllen und dort nicht auftauchen sollten [Sieben 2010: 441]. Um die Hilfsfunktionen zu verbergen („private") und gleichzeitig eine bessere Übersicht aller Funktionen zu erhalten, können in *Oracle* Packages eingesetzt werden. Der innere Aufbau von Packages erinnert stark an objektorientierte Programmiersprachen wie z.B. C++. In einem Spezifikationskopf werden alle öffentlichen („public") Funktionen deklariert, die der Benutzer u.a. auch in eigene Anwendungen einbauen kann. Im Funktionskörper („Body") folgen dann die Deklarationen der privaten Funktionen und der eigentliche Quellcode aller Funktionen.

Bei der *3DCityDB* werden in den öffentlichen Schnittstellen der Packages teilweise zusammengesetzte Datentypen definiert, damit die Übergabe von Variablen an die Funktionen etwas vereinfacht wird. Für das Erstellen von Instanzen dieses Datentyps können Konstruktoren definiert werden. Die Konstruktoren sind an den Datentyp gekapselt („Member-Functions") und keine externen Funktionen. Eine weitere Besonderheit ist das Erstellen von verschachtelten Tabellen. Das sind Datentypen, die in ihrer Struktur einem mehrdimensionalem Array entsprechen und sich demnach wie eine Tabelle verhalten. Der Umfang an Feldern ergibt sich in der Definition durch die Zuweisung eines zugesetzten Datentyps mit dem Befehl `IS TABLE OF` ***`datentyp`***. Ein anschauliches Skript-Beispiel befindet sich im Anhang A auf Seite 175.

Für die eben genannten Elemente existieren in *PostgreSQL/PostGIS* keine vergleichbaren Konzepte. Es gibt keine Packages, keine Member-Functions für benutzerdefinierte Datentypen und keine verschachtelten Tabellen. Laut etlichen Diskussionen in Mailing-Listen, sind die Entwickler von *PostgreSQL* der Meinung, diese objektorientierten Merkmale widerstrebten zu stark dem relationalen Standard der Datenbank und könnten auch anders gelöst werden. Der proprietäre *PostgreSQL Plus Advance Server* von EnterpriseDB bietet jene Features an, um die Migration zwischen den Systemen zu erleichtern [www27].

Die *PostgreSQL*-Dokumentation empfiehlt die Verwendung von verschiedenen Schemata, um Funktionen zu bündeln [www52]. Ein Schema ist in *PostgreSQL* ein gesonderter Namensraum mit eigenen Tabellen, Views, Sequenzen, Funktionen etc.. In Abbildung 13 ist zu erkennen, dass der Aufbau gegenüber dem public-Namensraum, der die Tabellen der *3DCityDB* und die *PostGIS*-Erweiterung beinhaltet, identisch ist. Alle Stored Procedures und dazugehörigen Komponenten wurden im Schema `geodb_pkg` angelegt und müssen stets über Punktnotation angesprochen werden (`geodb_pkg.objekt`). Ein Schema ist in *Oracle* gleichbedeutend mit dem Arbeitsbereich eines Benutzers. An dieser Stelle wird die Abweichung der Datenbankarchitektur von *Oracle* gegenüber *PostgreSQL* und auch gegenüber dem SQL Standard besonders deutlich. Das Datenbankschema und die PL/SQL-Packages sind an einen Benutzer gekoppelt. Ein anderer Benutzer kann über seine eigene, ggf. ältere oder aktuellere Version der *3DCityDB* verfügen und mit seinen Funktionen auf die Daten eines anderen Arbeitsbereiches zugreifen. Die Daten eines Stadtmodells können so über einen längeren Zeitraum in einem bestimmten Arbeitsbereich gespeichert sein, ohne dass Kompatibilitätsprobleme zu neueren Versionen der *3DCityDB* auftreten.

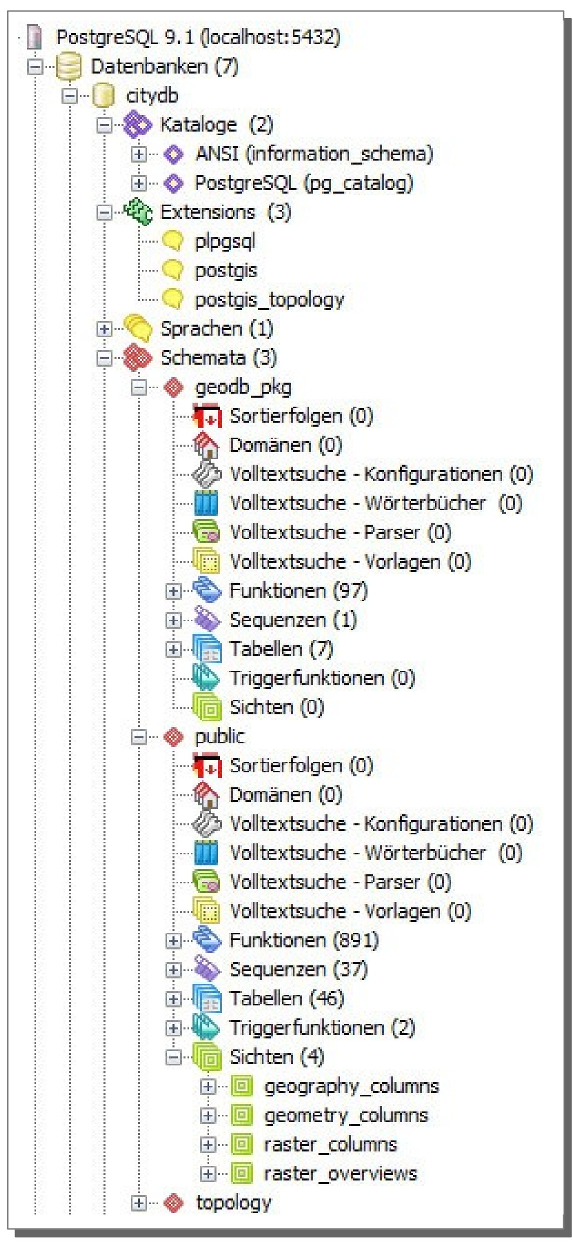


Abb. 13: Aufbau einer PostgreSQL Datenbank

Andere Benutzerschemata werden über Punktnotation angesprochen, wenn dafür Rechte vergeben sind. Die Arbeitsbereiche sind Teil einer Datenbank-Instanz, d.h. der installierten Softwareversion. Die eigentliche Datenbank befindet sich nur auf der Festplatte [vgl. Ahrends 2011a: 24f.]. In *PostgreSQL* sind die Benutzer entkoppelt von den Daten. Mehrere Benutzer können sich je nach vergebenen Privilegien einen Arbeitsbereich teilen, der in *PostgreSQL* auch als Datenbank bezeichnet wird. Auf den ersten Blick erscheint diese Terminologie etwas weniger abstrakt als jene in der Oracle-Welt. Personalisierte Funktionspakete verschiedener Versionen der *3DCityDB* würden hier über mehrere Schemata oder Datenbanken möglich sein. Insgesamt lässt sich das Fehlen von Packages in *PostgreSQL* entbehren.

Für die anderen nicht 1:1 übersetzbaren Teile wurden folgende Strategien eingesetzt:

- Die Konstruktoren wurden in eigenständige Funktion geschrieben.
- Die verschachtelten Tabellen sind als herkömmliche Tabellen definiert.

Da alle Funktionen innerhalb des `geodb_pkg` Schemas nicht gruppiert werden können, wurden den Funktionsnamen entsprechend ihres Package in der *Oracle*-Version ein Namens-Prefix gegeben, z.B. IDX-Package → `geodb_pkg.idx_*` etc.

6.2.2. Abfragen von Systemtabellen

Die Portierung einiger PL/SQL-Funktionen war nicht wegen der Syntax sondern durch die Abfrage von Systemtabellen eine Herausforderung, weil die gesuchten Werte in *PostgreSQL* bzw. *PostGIS* teilweise ganz anders verteilt waren als in *Oracle*. Wird z.B. ein Index in *Oracle* gelöscht, bleiben seine Metadaten in dem internen View `user_indexes` erhalten. Eine Statusabfrage auf eine entsprechende Spalte gibt den Wert `DROPPED` zurück. In *PostgreSQL* wird der Index komplett gelöscht. Eine Spalte in der Systemtabelle `pg_index` gib nur an, ob der Index korrekt und fehlerhaft konstruiert wurde. In der Tabelle `pg_index` ist es jedoch nicht möglich den Index anhand des Tabellen- und Spaltennamens zu spezifizieren. Dafür sind Joins mit den Tabellen `pg_attribute` und `pg_stat_user_indexes` notwendig (Beispiel: S.178).

Ein anderes Beispiel ist die Abfrage nach dem Typ eines Referenzsystems in der Tabelle `SDO_COORD_REF_SYS` in *Oracle Spatial*. Es existieren sogar mehrere Views für unterschiedliche Arten von Referenzsystemen, z.B. `SDO_CRS_GEOGRAPHIC3D` usw.. In *PostGIS* enthält ausschließlich die Tabelle `spatial_ref_sys` alle Daten zu Koordinatensystemen. Allerdings ist die Anzahl der Spalten stark begrenzt. Viele relevante Information befinden sich nur im Textfeld `srtext` und müssen mit unbequemen String-Methoden extrahiert werden (Beispiel: S.236).

An dieser Stelle sei noch einmal ausdrücklich empfohlen für ein besseres Verständnis die detaillierte Dokumentation der PL/SQL-Übersetzung im Anhang A.4. zu analysieren.

6.3. Umprogrammierung des Java-Clients Importer/Exporter

Der Titel des Kapitels deutet bereits darauf hin, dass der *Importer/Exporter* nicht für *PostGIS* erweitert, sondern als eigenständige Version umprogrammiert wurde. Vorschläge für eine flexiblere Softwarearchitektur werden abschließend in 6.3.4. gegeben. Zunächst wird der Aufbau und die Arbeitsweise des *Importer/Exporter*s kurz skizziert (6.3.1. & 6.3.2), damit der Leser den Umfang und die Auswirkungen der realisierten Code-Änderungen (6.3.3.) besser abschätzen kann. Das Ansprechen und Verarbeiten von Inhalten einer *PostGIS*-Datenbank mittels JDBC ist vergleichsweise schlecht dokumentiert. Anhand der Kommentare in den Klassen-Dokumentationen (API) von

PostGIS und *PostgreSQL* [www53, www54] ist erkennbar, dass die letzten inhaltlichen Aktualisierungen etliche Jahre zurückreichen. Direktvergleiche zu JDBC-Workflows mit *Oracle Spatial* existieren im Gegensatz zu den Migrationsvorschriften von SQL- und PL/SQL-Skripten nicht.

6.3.1. Wie arbeitet der Importer/Exporter?

Der *Importer/Exporter* wurde mit dem Ziel entwickelt selbst größte CityGML-Instanzdokumente (> 4 GB) so performant wie möglich in die Datenbank zu laden. Die technische Umsetzung ist sehr gut und detailliert in [Nagel, Stadler 2008] beschrieben. Die folgende Absätze geben nur eine Kurzform der wichtigsten Inhalte wieder.

Wie bereits in 3.2. erwähnt, wurden die Java-Frameworks SAX und JAXB verwendet. Die Programmierschnittstellen von JAXB übernehmen das eigentliche Validieren und Prozessieren des CityGML-Dokuments auf ein Java-Objektmodell und wieder zurück (als „Unmarshalling" bzw. „Marshalling" bezeichnet). Mit JAXB kann der Entwickler diese Bindung zwischen Objekten und XML-Inhalten optimal konfigurieren, wodurch die Struktur des Ausgangsdokumentes besser und vor allem einfacher abgebildet werden kann als mit der viel allgemeineren ebenfalls modellbasierten Verarbeitung des XML Document Object Models (DOM). Aufgrund des referentiellen und rekursiven Aufbaus von CityGML sind streambasierte Verfahren wie SAX für eine objekt-orientierte Abbildung äußerst ungeeignet, weil sie beim sequentiellen Lesen („Parsen") der XML-Bäume nur die physikalische nicht aber deren logische Struktur berücksichtigen [vgl. Scholz, Niedermeyer 2009]. JAXB hat jedoch einen entscheidenden Nachteil: Das XML-Dokument muss für die Verarbeitung komplett in den Hauptspeicher geladen werden, was bei vielen CityGML-Dateien nicht durchzuführen wäre. Dieses Problem wurde umgangen, indem das XML-Dokument während der Prozessierung in kleinste Datenblöcke („Chunks") unterteilt wird. Die Zerlegung kann wiederum durch SAX gesteuert werden, weil dessen ereignisorientierte Arbeitsweise das Auslösen entsprechender benutzerdefinierter Befehle ermöglicht, z.B. wenn der aufgezeichnete Speicherverbrauch einen bestimmten Wert erreicht. Alle zu einem Chunk gehörenden SAX-Ereignisse werden in einem Zwischenspeicher aufgezeichnet und von dort mit JAXB weiterverarbeitet. Danach wird der Zwischenspeicher wieder frei gegeben. Der Arbeitsspeicherverbrauch hängt also nur von der parallel in verschiedenen Zwischenspeichern gehaltenen Datenmenge ab und nicht von der Ausgangsgröße des Dokumentes.

Die parallele Verarbeitung der eben beschriebenen Prozesse wird durch softwareseitiges Multithreading realisiert. Durch die Nebenläufigkeit von Verarbeitungssträngen („Threads") kann die Ausführungsgeschwindigkeit eines Programms gesteigert werden [vgl. Ullenboom 934ff.]. Weil jedoch viele Threads ebenfalls einen erhöhten administrativen Aufwand für Prozessor und Arbeitsspeicher darstellen, werden Threads mit gleichen Aufgaben in Threadpools organisiert und bleiben nach ihrem Durchlauf zur Wiederverwendung erhalten. Damit werden Verzögerungen bei der Erzeugung neuer Threads vermieden. Die Threadpools wurden als Warteschlange („quene") modelliert,

damit die mit ihm assoziierten konkurrierenden Threads synchronisiert auf die abzuarbeitenden Aufgaben zugreifen.

Durch die Zerteilung des CityGML-Dokumentes ergibt sich jedoch ein weiteres Problem. XLink-Referenzen für Relationen zwischen Geometrien oder Attributen, die weit auseinander liegende Features im Dokument miteinander verknüpfen, werden getrennt und können theoretisch nicht mehr korrekt in der Datenbank abgebildet werden. Die Lösung war das Speichern von XLinks (Element + GMLID) in eine temporäre Tabelle der Datenbank, deren Einträge sich nach dem Import aller Objekte anhand der global eindeutigen GMLIDs auflösen lassen. Abbildung 14 skizziert grob die in diesem Unterpunkt beschrieben Prozesse. Im Anhang findet sich aus Seite 295 eine detailliertere Fassung aus [Nagel, Stadler 2008].

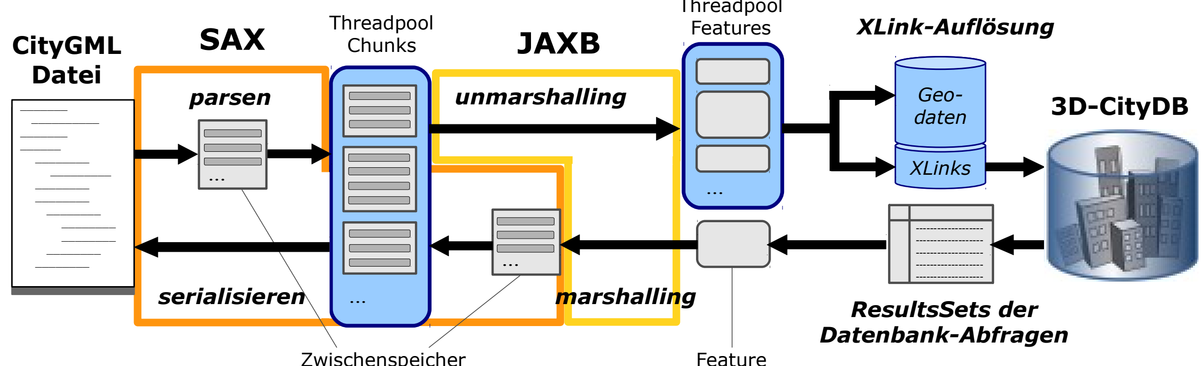


Abb. 14: Prozessketten beim Im- und Export von CityGML-Daten [vgl. Nagel, Stadler 2008]

JDBC als Verbindungsschnittstelle zwischen Java und Datenbank wurde bereits einige Male erwähnt. Sie bildet seit vielen Jahren den konventionellen Weg zur Realisierung von datenbankgestützten Java-Anwendungen und ist entsprechend gut im Internet und in zahlreichen Büchern dokumentiert [vgl. u.a. Abst 2010]. Die Zugriffsmethoden vom JDBC sind datenbankneutral, die Anwendung damit unabhängig von der Wahl des DBMS. Bei der Verarbeitung von komplexen Datentypen wie Geometrien müssen aber Klassen spezifischer Treiber der Datenbankhersteller mit eingebunden werden.

Zentrale JDBC-Komponenten sind die `Statement`-Klassen, welche Datenbankabfragen in Java beschreiben, und das `ResultSet`, welches die Ergebnisse von Abfragen speichert. Der *Importer/Exporter* arbeitet vorrangig mit `PreparedStatements`. Das sind vor-kompilierte SQL-Befehle. Bei mehrfachem Ausführen muss das DBMS nicht mehr die Grammatik des SQL-Strings prüfen, selbst wenn einzelne Werte ausgetauscht werden. Das Abrufen der Stored Procedures übernehmen `CallableStatement`-Objekte. Durch die Verwendung eines generischen Connection Pools wie dem *UCP* reduziert sich der Einsatz von Klassen DBMS-spezifischer JDBC-Treiber auf ein Minimum und die Anwendung kann sich ohne viele Anpassungen mit verschiedenen DBMS verbinden.

6.3.2. Projektaufbau in der Entwicklungsumgebung Eclipse

Wie viele Java-Anwendungsprogramme wird der *Importer/Exporter* in der Open-Source-Entwicklungsumgebung *Eclipse* programmiert. Das zentrale Code-Repository wird mit dem Projektmanagement System von ChiliProject verwaltet [www55]. Mit dem *Eclipse* Plugin *SubClipse* von Tigris kann der Quellcode des *Importer/Exporter* in verschiedenen Versionen vorgehalten werden. Der aktuellste Stand befindet sich im Ordner „Trunk“, von dem zu Beginn der Portierung eine lokale Kopie ausgespielt wurde („Checkout“). Nach erfolgreicher Umsetzung wurde für die *PostGIS*-Version ein eigener Projektordner („Branch“) im Repository angelegt, der weiterhin mit dem Trunk-Ordner und anderen Branches synchronisiert werden kann („merge“).

Der interne Aufbau des *Importer/Exporter* ist modular gestaltet und ermöglicht damit die Erweiterung durch Plugins. Diese Idee kennt der Leser bereits am Beispiel von CityGML. Das Abbilden der Inhalte einer CityGML-Datei in Java, die Verwaltung der partiellen Datenströme (I/O) und auftretender Ereignisse sowie der Aufbau und die Bedienung der Benutzeroberfläche (GUI - Graphical User Interface) sind datenbankunabhängig und mussten für die Portierung nicht beachtet werden.

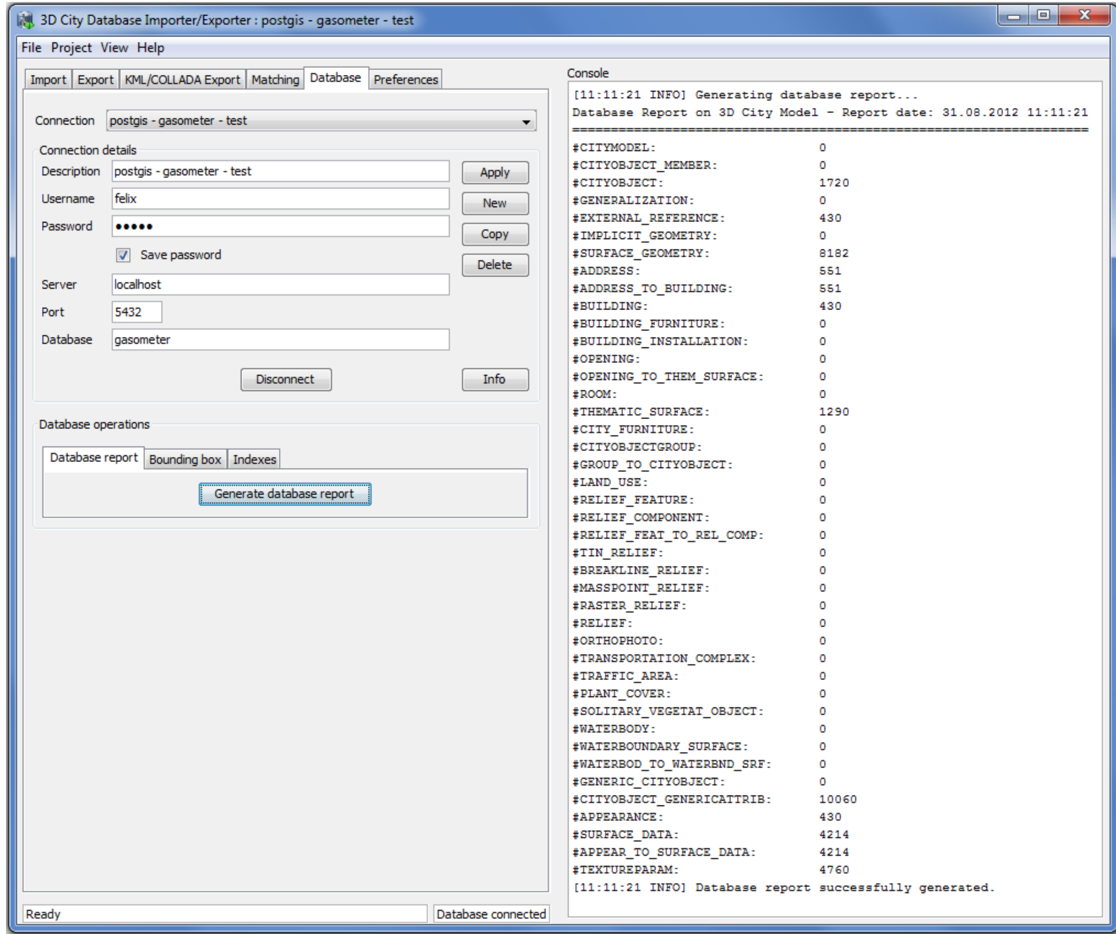


Abb. 15: GUI des Importer/Exporter

Die wesentlichen Änderungen mussten in den Funktionsmodulen des CityGML-Imports und -Exports und in einigen Datenbank-Controller-Klassen vorgenommen werden. Für den KML-Export wären natürlich ebenso Eingriffe notwendig. In das Bibliotheks-Verzeichnis müssen die JDBC-Treiber von *PostgreSQL* und *PostGIS* eingeladen werden, um von *Eclipse* interpretiert zu werden. Abbildung 15 zeigt die GUI des Datenbank-Panels vom *Importer/Exporter*. Für Im- und Exporte lassen sich inhaltliche und räumliche Filter auf die jeweilige Datenquelle anwenden. Die Bounding Box kann intuitiv über ein Kartenfenster („widget") überprüft oder ggf. neu aufgezogen werden. Ein Konsolen-Fenster zeichnet Log-Einträge auf. Für das Abbildungsbeispiel wurde zuvor ein Datenbank-Bericht angefordert, der die Einträge in den Tabellen zählt. Weitere Abbildungen befinden sich im Anhang D.1..

6.3.3. Die wichtigsten Änderungen im Quellcode

Nachdem der grundlegende Aufbau der Anwendung analysiert war, wurde in den Java-Klassen systematisch nach Importen von Paketen aus *Oracle*-spezifischen Bibliotheken gesucht, weil dies in erster Linie als Hauptindikator für Datenbank-spezifischen Quellcode galt. Die Importe wurden auskommentiert, so dass sich die Java-Klassen nicht mehr kompilieren ließen. Wie in jeder modernen Entwicklungsumgebung wurden automatisch die fehlerhaften Zeilen mit den unreferenzierten Objekten markiert. Damit war klar, welche Quellcode-Passagen verändert werden mussten. Diese einfache Strategie vernachlässigte zwar vorerst ein allgemeines Verständnis aller Software- bzw. auch Klassenbestandteile, führte jedoch viel schneller zum Erfolg als eingangs geplant (5 statt der angedachten 12 Wochen Bearbeitungszeit). Da die *PostGIS*-Version aber über diesen Meilenstein hinaus parallel zur Masterarbeit weiterentwickelte wurde und wird, hat sich auch der Blick des Autors mittlerweile auf die allgemeineren Funktionsbereiche ausgeweitet. Die folgenden Unterpunkte fassen die wichtigsten Änderungen thematisch gruppiert zusammen. Für explizite Quellcode-Beispiele sei auf Anhang B verwiesen.

Datenbankgeometrien in Java

Die Stadtmodell-Geometrien sind nicht nur die komplexesten Daten, die aus der *3DCityDB* in die Java-Ebene und umgekehrt geladen werden, durch ihre unterschiedliche Struktur in *Oracle Spatial* und in *PostGIS* war die Portierung ihrer Methoden auch am anspruchsvoll-sten gegenüber den restlichen Umsetzungen.

In Java werden SDO_GEOMETRY-Objekte mit der Klasse `JGeometry` beschrieben. `JGeometry` bietet für alle Arten von Geometrietypen Konstruktoren an, die jeweils mit einem Array der Koordinaten und INTEGER-Werten zu der Dimension und dem SRID instanziiert werden. Die Methoden sind vielfältig [www56]. Mit `getElemInfo()` und `getOrdinateArray()` lassen sich die entsprechenden Attribute von SDO_GEOMETRY in separate Arrays speichern. Die Klassen des CityGML-Exports machen davon intensiv Gebrauch, um Geometrien in Java zu rekonstruieren.

Bei *PostGIS* werden Geometrien in Java wie schon auf Datenbankebene hauptsächlich mit Strings erfasst. So gibt zum Beispiel die abstrakte Klasse `Geometry` bei der Methode `getValue()` einen String zurück. Konstruiert wird eine Instanz von `Geometry` über die Methode `geomFromString(String)` der Klasse `PGgeometry`, auf die gleich noch ausführlicher eingegangen wird. Im Gegensatz zu der zentralen `JGeometry`-Klasse für *Oracle Spatial* umfasst der JDBC-Treiber von *PostGIS* eigene Klassen für alle geometrischen Ausprägungen, die von `Geometry.composedGeom` erben. Wo bei *Oracle* der Kontrollfluss der Geometrie-Prozessierung über das `SDO_ELEM_INFO-ARRAY` gesteuert werden kann, müssen bei *PostGIS* die Methoden dieser Unterklassen verwendet werden. Anschauliche Beispiele finden sich in Anhang B ab Seite 252. Bevor die eben beschriebenen Geometrieobjekte mit der `setObject()`-Methode an `PreparedStatemets` für die Datenbank übergeben werden können, muss ihre spezifische Struktur mit einem allgemeineren JDBC-Objekt umhüllt werden, auch „wrapping" genannt. Die Wrapper-Klasse für `JGeometry` ist `STRUCT`, bei *PostGIS* ist es `PGgeometry`. Sie werden auch beim umgekehrten Weg eingesetzt, wenn Daten aus den `ResultSets` der Datenbankabfragen extrahiert werden.

Die folgenden Abbildungen zeigen die wichtigsten Java-Verarbeitungsschritte für die Geometrieobjekte auf ihrem Weg vom Dokument zur Datenbank und zurück. Die roten Pfeile symbolisieren Iterationen. Spezifischen Methoden der Klassen sind in kursiven Lettern dargestellt. Auf *Oracle*-Seite wird nahezu alles über `JGeometry` gesteuert. Bei der *PostGIS*-Variante ist `PGgeometry` nicht nur ein Wrapper sondern auch Konstruktor für `Geometry`. Für den Export speichern die Zugriffsmethoden („getter") der `Geometry`-Klassen die Werte der `ResultsSets` ohne ein zusätzliches Array wie bei der Oracle-Version direkt in die Listen für die CityGML-Primitive zurück. Die getter- und setter-Methoden für die CityGML-Datei verwenden hauptsächlich Klassen der *citygml4j*-Bibliothek.

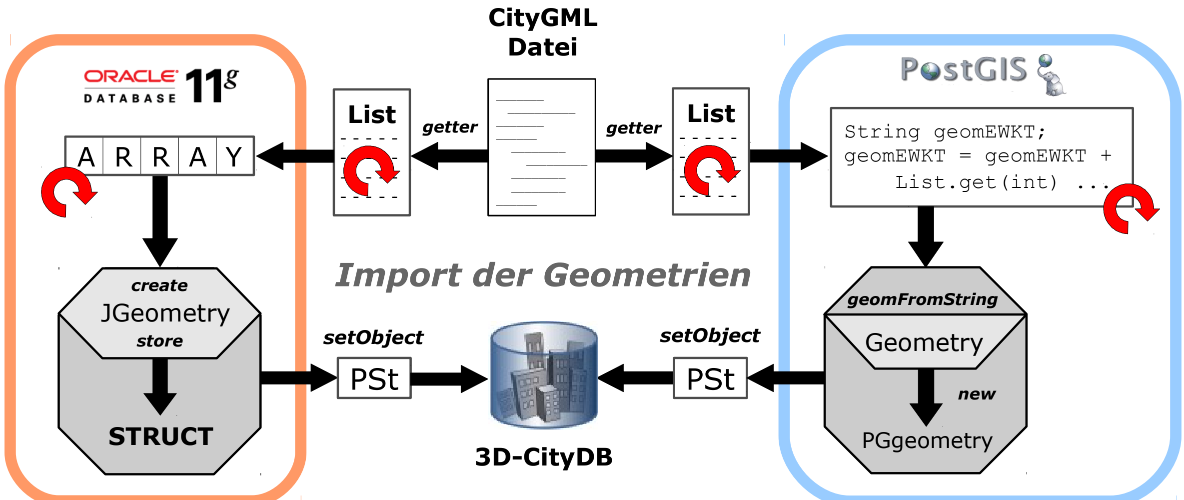


Abb. 16: Importweg der Geometrien in Java (PSt = PreparedStatement)

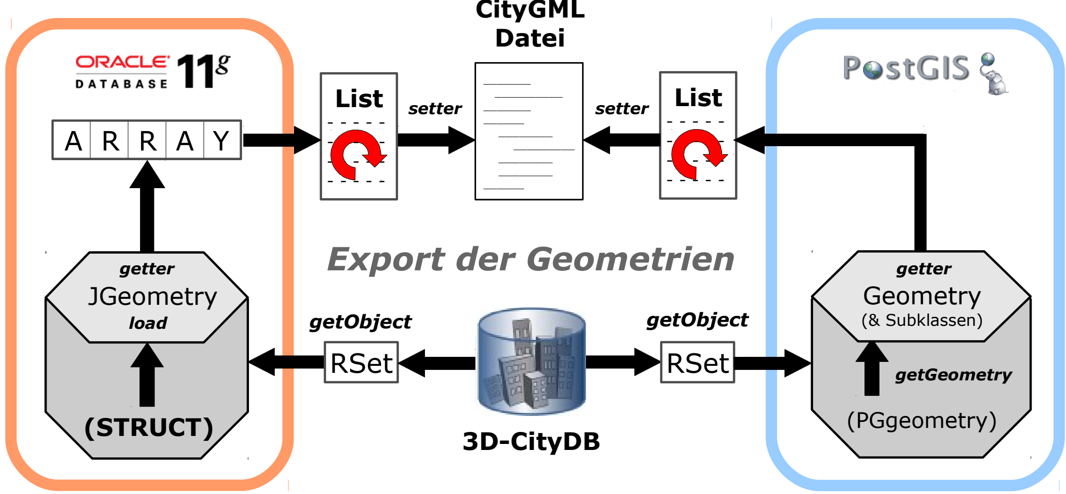


Abb. 17: Exportweg der Geometrien in Java (RSet = ResultSet)

Texturen

Die Texturdateien sind physikalisch kein Teil eines CityGML-Dokuments, sondern liegen für gewöhnlich in einem separaten Ordner im selben Verzeichnis vor. Über die Bildadresse (URI) werden sie mit einer CityGML-Appearance assoziiert (XLink). Der Import der Texturen folgt erst nach der Auflösung eines entsprechenden XLinks, indem ein `UPDATE` auf die Tabelle `Surface_Data` ausgeführt wird. Für den ORDImage-Datentyp funktioniert das nicht direkt. Erst muss das ORDImage für die jeweilige Tabellen-Zelle initialisiert und mit einem `SELECT FOR UPDATE`-SQL-Befehl von parallelen Prozessen isoliert werden. Aus dem `ResultSet` dieser Abfrage wird ein noch leeres ORDImage-Objekt mit der Methode `getORAData-Factory` für die Befüllung mit Binärdaten aus einen `InputStream` zur Verfügung gestellt. Das fertige ORDImage in Java kann dann mit `setORAData` im eigentlichen `UPDATE`-Statement für die Datenbank übergeben werden. Da der *PostgreSQL*-Datentyp BYTEA eine reine Ansammlung von Binärdaten ist, werden die im `InputStream` eingelesenen Werte direkt mit `setBinaryStream` im Statement gesetzt. Es sind also deutlich weniger Schritte notwendig.

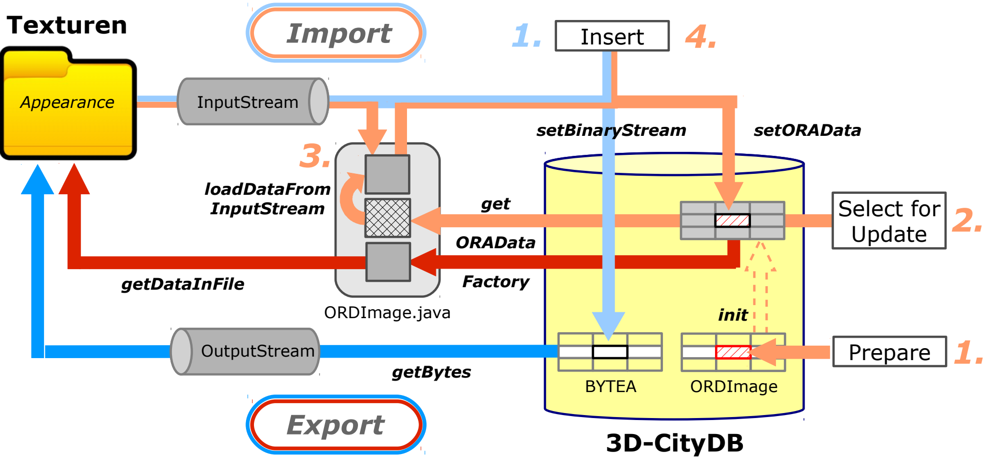


Abb. 18: Im- und Export von Texturen

Der Export von Texturen umfasst bei *Oracle* dank des Zusammenspiels der ORDImage-Java-Klasse mit seinem Datenbank-Pendant ebenso wenig Code-Zeilen wie das Schreiben der BYTEA-Daten in Bilddateien bei der *PostgreSQL/PostGIS*-Variante. Abbildung 18 veranschaulicht nochmal die unterschiedlichen Herangehensweise beider Systeme. Sollten für die *Oracle*-Version der *3DCityDB* BLOBs für Texturen eingesetzt werden, dürfte sich dies sehr ähnlich zum *PostgreSQL*-Weg gestalten.

Unterschiede bei Datentypen und Funktionen

Wie bei den Datentypen gibt es bei beiden DBMS Funktionen, die unterschiedliche Namen, ggf. sogar einen abweichende Aufbau haben, aber die gleiche Aufgabe ausführen. Tabelle 4 gibt eine kurze Übersicht der aufgetretenen Fälle.

Tab. 4: Unterschiedliche Namen, gleiche Funktion

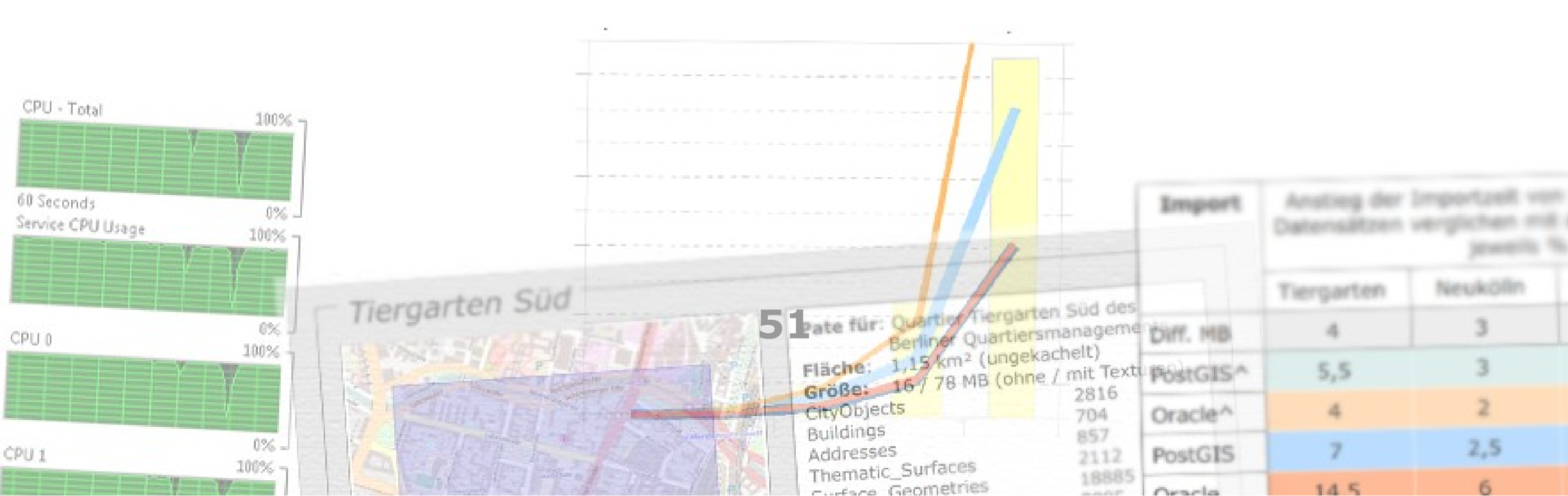
Oracle	PostgreSQL/PostGIS	Erklärung
`SYSDATE`	`now()`	gibt Datum und Systemzeit aus
`id_SEQ.nextval`	`nextval('id_SEQ')`	gibt nächsten Wert einer Sequenz zurück
`nvl(arg, 0)`	`COALESCE(arg, 0)`	falls `arg` einen leeren Wert hat (`null`), wird er durch den zweiten Wert ersetzt, hier 0
`sdo_aggr_mbr(geom)`	`ST_Extent(geom)`	errechnet die Bounding Box
`SDO_CS.TRANSFORM (geom)`	`ST_Transform(geom)`	transformiert die Koordinaten der übergebenen Geometrie
`SDO_RELATE (geom1,geom2)`	`ST_Relate (geom1,geom2)`	prüft die topologische Beziehung zwischen beiden Geometrien

Auf Java-Ebene sind diese Funktionen allerdings in String-Variablen verpackt und haben damit keinen Einfluss auf das erfolgreiche Kompilieren der Klassen. Einige notwendige Konvertierungen fielen erst durch Kompilierungsfehler innerhalb des DBMS auf, die in der Konsole des *Importer/Exporter* als eine Fehlermeldung („Exception") des *PostgreSQL*-JDBC-Treibers (`PSQL-Exception`) erschienen und damit lokalisiert werden konnten.

6.3.4. Zukünftige Struktur der Software

Ungeachtet der funktionellen Änderungen wäre es langfristig gesehen eleganter die Software so zu strukturieren, dass beide *Importer/Exporter*-Varianten in einer Version gebündelt werden. Klassen mit spezifischen Ausprägungen müssten dann gegen ein gemeinsames Interface programmiert werden. In diesem Zuge wären weiter Versionen für andere räumliche RDBMS denkbar. Im Datenbank-Panel des *Importer/Exporter* (siehe Abblidung 15) würde der entsprechende Datenbank-Treiber auswählbar sein. Nach dem Verbindungsaufbau würde dann die Implementierung der jeweiligen spezifischen Klassen gegen ihre Interfaces gemäß dem gewählten Treiber vorgeschrieben werden. Die Programmierung einer solchen Architektur ist jedoch gerade bei stark abweichenden Konzepten, wie sie hier zum Teil demonstriert wurden, mit einem größeren Aufwand verbunden.

Teil III

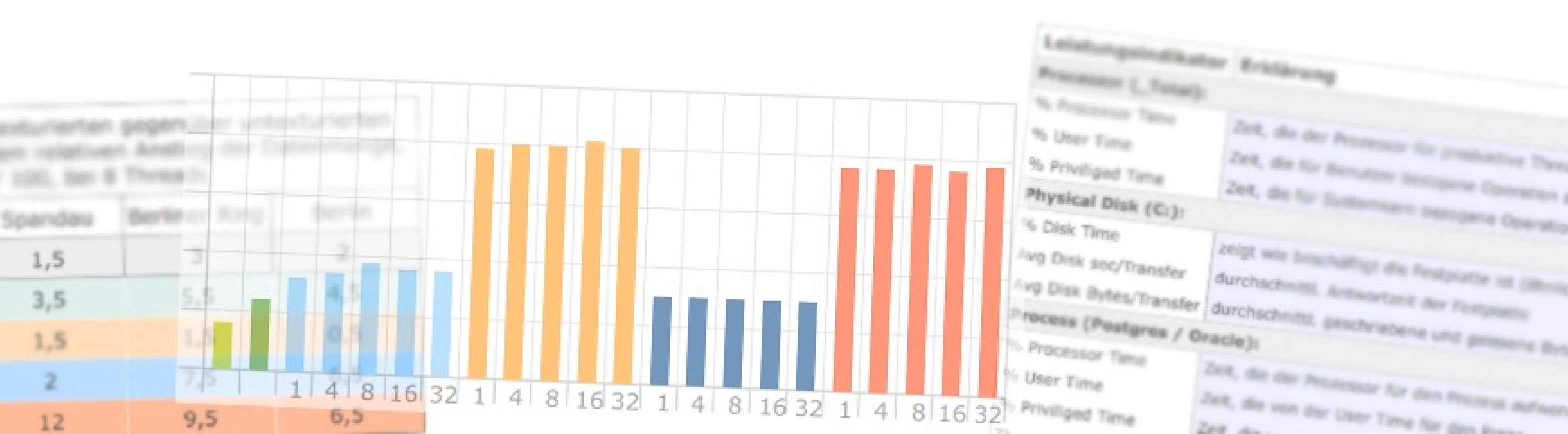


Performanz-Vergleich beider Versionen

Was erwartet den Leser?

Nachdem die in Kapitel 5 festgelegten Teile der *3DCityDB* und des *Importer/Exporter* erfolgreich für *PostGIS* portiert werden konnten, war es spannend zu untersuchen, wie gut die Umsetzung gegenüber dem Original funktioniert? Dazu wurde eine Vergleichsanalyse konzipiert und parallel ein Monitoring der Rechnerauslastung durchgeführt.

Die Vorgehensweise bei den Untersuchungen wird Kapitel 7 erklärt. Die Präsentation der Ergebnisse folgt in Kapitel 8, ihre Interpretation in Kapitel 9. Dieses letzte Kapitel schließt den Bogen zu den Ausgangsfragen. Das Wissen aus dieser Masterarbeit soll für seine weitere Verwendung, ob als Basis, Schnittstelle oder Leitfaden, auf zukünftige Arbeiten projiziert werden.



Kapitel 7

Konfiguration und Performance-Monitoring

Die Entwicklungen der *3DCityDB* und des *Importer/Exporter* wurden stets von den Bestrebungen nach einer optimalen Performanz begleitet. Viele Rezepte, die in Fachbüchern („Cookbooks") zur Datenbank-Administration (DBA) und Applikationsentwicklung beschrieben sind, finden sich in den Umsetzungen des Projektes wieder (siehe 7.1.). Die Datenbank-Lösung für CityGML in *PostGIS* konnte durch Portierung dieser universellen Stärken erheblich profitieren. Neben dem Datenbank-Design kann aber auch die Konfiguration des DBMS auf die eigenen Hardwareressourcen ein entscheidender Faktor für die Performanz sein. In 7.2. wird darauf näher eingegangen. Wenn alle Bestandteile ihren Zweck erfüllen, folgen in der Regel zahlreiche Testphasen, um Fehler oder Performanz-Engpässe („Flaschenhälse") zu ermitteln. In 7.3. wird erläutert nach welchem Schema und mit welchen Methoden die praktischen Umsetzungen der Masterarbeit geprüft worden sind.

7.1. Best Practices

Viele der eingesetzten Praktiken zur Verbesserung der Performanz folgen allgemein anerkannten Empfehlungen („Best Practices") aus Online-Dokumentationen und Cookbooks (siehe u.a. Fiorillo 2012). Die meisten wurden bereits in den vergangenen Kapiteln erwähnt, aber nicht immer in diesem Sinne definiert. Für einen besseren Überblick sind sie an dieser Stelle nochmals mit einer Kurzbeschreibung aufgelistet:

Datenbank-Schema:

- Vereinfachung des UML-Datenmodells, um die Zahl zeitaufwändiger Tabellen-Joins zu verringern
- Normalisierung der Daten in einem RDBMS für eine redundanzfreie, persistente wie konsistente Haltung der Daten
- Einsatz von Spalten-Indizes (auch für Geometriespalten)

PL/SQL-Skripte:

- Verwendung von Packages zu besseren Strukturierung der Funktionen
- Einsatz von dynamischem SQL, d.h. Abfragen werden zur Laufzeit erstellt und können sich währenddessen ändern. Dies ist meist in Verbindungen mit Datenbank-Applikationen notwendig. Ausgeführt werden sie als String in einem `EXECUTE IMMEDIATE`-Befehl. Datenbanken puffern diese Strings, damit sie beim wiederholten Ausführen nicht mehr interpretiert werden müssen. Variablen werden durch Platzhalter ersetzt (bei Oracle `:1`, bei PostgreSQL `$1`) und nach dem String mit `USING` an die Abfrage gebunden. Sie heißen deshalb Bind-Variablen. Ein exemplarisches Beispiel befindet sich auf Seite 40.

JDBC-Schnittstelle:

- Verwendung von `PreparedStatements`, welche vorkompilierte SQL-Befehle mit austauschbaren Werten an die Datenbank übergeben, quasi dynamisches SQL auf Applikationsseite.
- Batch-Processing, d.h. `Statements` werden in einem Batch-Objekt gesammelt und in einem Zug an die Datenbank geschickt, damit nur einmal eine Verbindung aufgebaut wird und nicht bei jeder einzelnen Abfrage.
- Implementierung eines Connection Pools zum Vorhalten von Verbindungen

Java:

- Nebenläufigkeit der Software-Prozesse (Multithreading)
- modularer Aufbau von Packages und Klassen für eine bessere Erweiterbarkeit

7.2. Konfiguration des DBMS

Damit neue Software nach der Standardinstallation auch auf deutlich älteren Maschinen lauffähig ist, müssen die Parameter zur Ressourcenauslastung auf einem vergleichsweise niedrigeren Level gesetzt sein. Eine optimale Konfiguration an die eigene Hardware kann wie bei *Oracle* automatisch durch die Software selbst erfolgen oder sie muss manuell editiert werden, was bei *PostgreSQL* der Fall ist. Für die Vergleichsanalyse war vorgesehen, mehrere Test jeweils mit der Standardinstallation durchzuführen und danach einen Vergleich mit dem jeweils „getunten" DBMS zu suchen.

7.2.1. Das sich selbst verwaltende Oracle

[Fröhlich 2009: 519] beschreibt *Oracle* sehr passend als eine sich selbst verwaltende Datenbank. Mit jeder neuen Version versucht Oracle dem Datenbank-Administrator Arbeit abzunehmen, indem neue Features geschaffen werden, dank denen sich die Software weitestgehend selbstständig an den Workload anpasst. Wer also nach einem einfachen Konfigurationsleitfaden sucht, findet sich in vielen Quellen oft im „Troubleshooting" wieder, d.h. Tuning der Datenbank aufgrund eines erkannten Problems und nicht etwa

zum Zwecke des Tunings selbst [Ahrends et al. 2011a: 843]. Die Zuweisung von Arbeitsspeicher (RAM) übernimmt das *Automatic Memory Management (AMM)*. Das *Automatic Shared Memory Management (ASMM)* verteilt die zugewiesenen Ressourcen auf verschiedene Komponenten. Ohne manuelle Reglementierung nutzt *Oracle* die Systemressourcen zwar voll aus, belegt allerdings auch einen beträchtlichen Teil im RAM. Über den *Enterprise Manager*, einer Administrationsoberfläche in einem Web-Browser, kann das AMM verändert werden. Es lässt sich außerdem eine Liste aller Initialisierungsparameter begutachten, an der sehr gut zu erkennen ist, wie viele Einstellungen dynamisch Werte besitzen und von welchen Softwareteilen sie verwaltet werden. Aufgrund dieses hohen Maßes an Selbstverwaltung wurde entschieden, keine konfigurierte Variante von *Oracle* in den Datenbank-Vergleich mit einzubeziehen, sondern die Default-Installation als „bereits getunt“ zu betrachten.

7.2.2. postgresql.conf

In der Literatur zu *PostgreSQL* wird immer wieder dazu aufgerufen, die sehr niedrig angesetzten Grundeinstellungen in der Konfigurationsdatei postgresql.conf anzupassen, bevor Aussagen über die Schnelligkeit des Datenbank-Servers getätigt werden. Anhand der Angaben von [Eisentraut 2011] und [Smith 2010], zwei Standardwerken unter den *PostgreSQL*-Fachbüchern, und einem aktuellen Benchmarking-Artikel aus der ADMIN-Zeitschrift [vgl. Brendel 2011] wurden folgende Parameter editiert:

Tab. 5: Änderungen in postgresql.conf

Parameter	editiert	vorher	Anmerkungen
`shared_buffers`	512MB	32MB	Faustregel: ¼ des RAMs, aufgrund der 32Bit-Installation ist nur max. 1 GB möglich
`temp_buffers`	16MB	8MB	bei vielen temporären Tabellen vorteilhaft
`work_mem`	12MB	1MB	Faustregel: RAM / `max_connections` / 4 u.a. für Joins verwendet
`maintenance_work_mem`	512MB	16MB	Faustregel: 1/8 des RAMs u.a. für DDL-Statements verwendet
`wal_buffers`	16	-1	nicht an `shared_buffers` binden
`checkpoint_segments`	128	3	definiert die Zeitspanne zum nächsten Absetzen eines Checkpoints
`checkpoint_timeout`	30min	5min	
`random_page_cost`	2.0	4.0	bevorzugt die Indizes stärker
`effective_cache_size`	4GB	128MB	Faustregel: RAM minus `shared_buffers`; ist nur eine Angabe für den SQL-Planer
`log_rotation_size`	0	10MB	Beginn von neuer Log-Datei nur nach Tagen
`log_min_duration_statement`	100ms	-1	alle Abfragen, die über zwei Minuten brauchen, werden geloggt
`log_duration`	(on)	off	registriert die Dauer aller SQL-Befehle, nur sinnvoll bei automatisierter Log-Auswertung
`log_line_prefix`	`'%t [%p]: [%l-1] '`	`'%t '`	Formatvorlage für Log-Einträge, die sich mit *pgFouine* analysieren lassen

7.3. Überwachungsmethoden der DBMS-Performanz

Die Möglichkeiten die Performanz einer Datenbank zu analysieren sind so vielfältig, dass eine Masterarbeit sich damit ausschließlich befassen könnte. Sie werden im letzten Unterpunkt dieses Kapitels besprochen (7.3.2). Im Rahmen dieser Arbeit beschränkten sich die Untersuchungen fast ausschließlich auf einen Vergleich der Ausführungszeiten von Datenim- und Exporten der beiden *Importer/Exporter*-Versionen. Dabei sollte auch überprüft werden, wie ausgewogen die Leistung beider DBMS mit wachsender Datensatzgröße und steigender Anzahl an parallelen Ausführungssträngen skaliert wird. Parallel wurde die Beanspruchung der Hardware überwacht. Ein professionelles Troubleshooting und SQL-Plananalysen mit dem `EXPLAIN`-Befehl war aus zeitlichen Gründen nicht durchführbar.

7.3.1. Versuchsaufbau

Durch die Log-Ausgabe im Konsolenfenster des *Importer/Exporter* kann sekundengenau bestimmt werden, wann ein Prozess initialisiert und wann er vollständig abgeschlossen wurde. Für die Anzahl an parallelen Ausführungsthreads wurden fünf Abstufungen festgelegt (1, 4, 8, 16 und 32), die jeweils vor dem Im- oder Export in den Programmoptionen eingestellt werden mussten. Es wurden fünf unterschiedlich große Datensätze in zweifacher Ausführung aus einer vorhandenen *Oracle*-Datenbank mit dem kompletten 3D-Stadtmodell von Berlin exportiert – einmal mit Texturen, einmal ohne. Die untexturierten Datensätze beinhalten zudem keine Appearance-Objekte. Die Auswahl der Datensätze sollte sich an denkbaren Untersuchungsgebieten für Raumanalysen in einem Stadtmodell orientieren. Da die Anzahl an Texturen das Limit für die im Windows-Dateisystem (FAT32) zugelassene Menge an Dateien in einem Ordner schnell überschreiten kann, wurden die Datensätze bei den drei größten Beispielen in mehrere Ordner gekachelt exportiert.

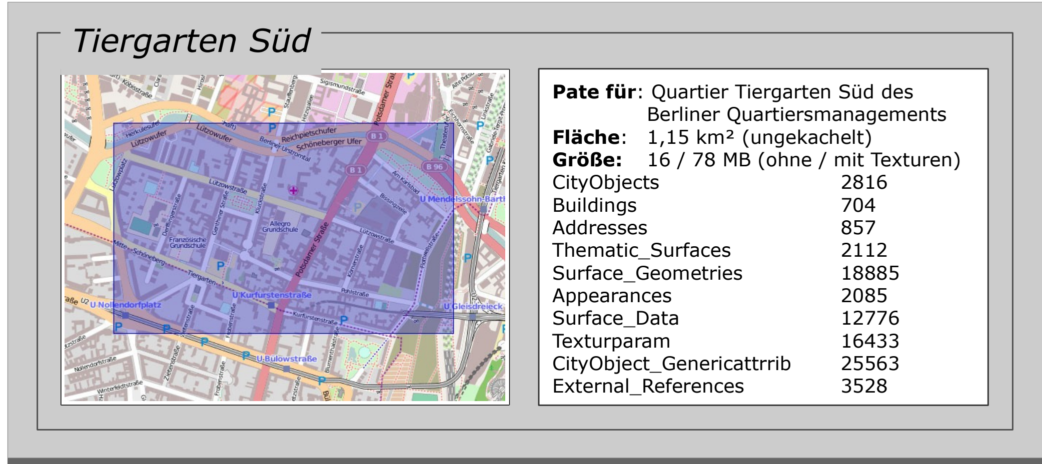
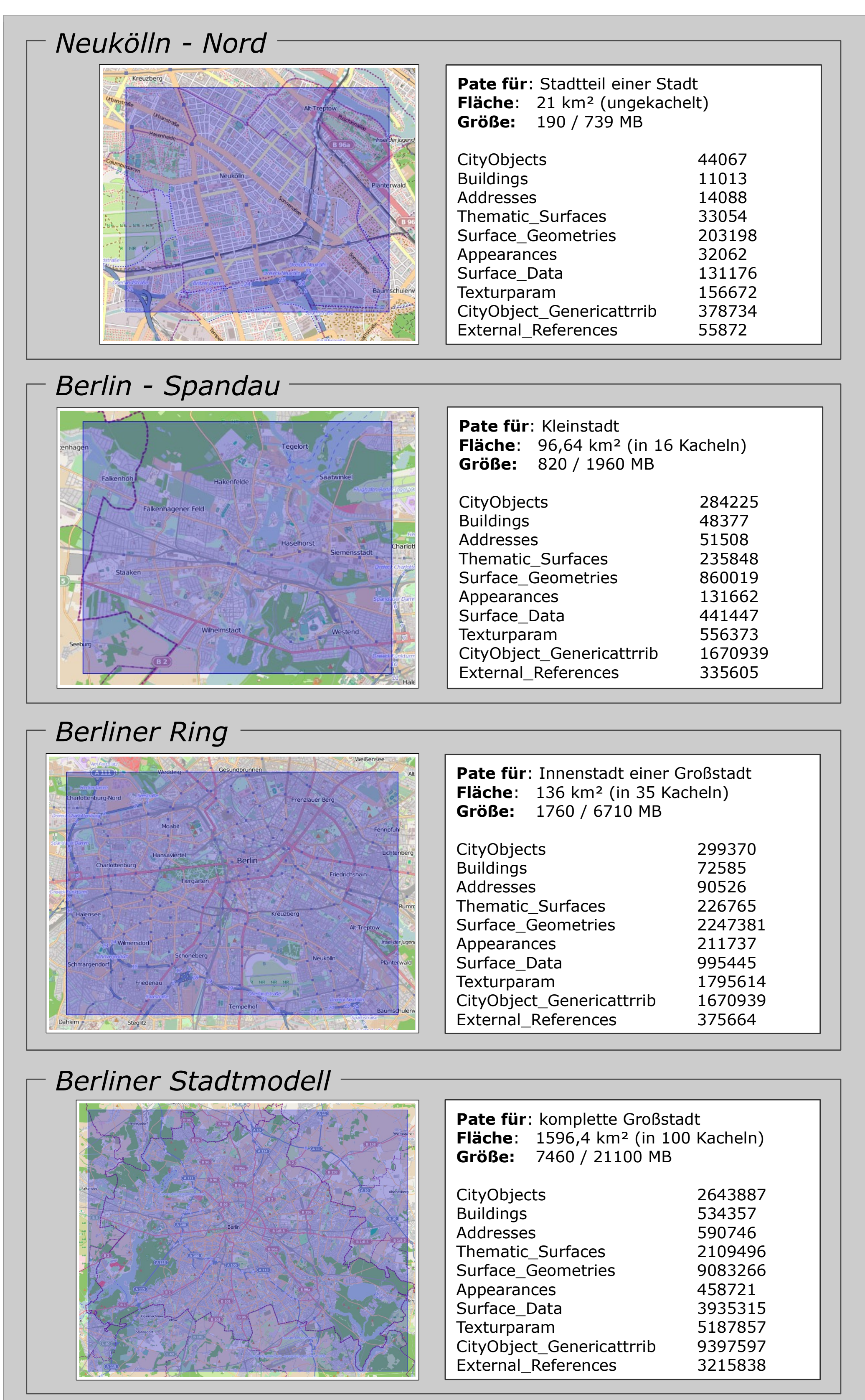


Abb. 19a: Ausgewählte Datensätze im Überblick (Fortsetzung auf nächster Seite)



Neukölln - Nord

Pate für: Stadtteil einer Stadt
Fläche: 21 km² (ungekachelt)
Größe: 190 / 739 MB

CityObjects	44067
Buildings	11013
Addresses	14088
Thematic_Surfaces	33054
Surface_Geometries	203198
Appearances	32062
Surface_Data	131176
Texturparam	156672
CityObject_Genericattrrib	378734
External_References	55872

Berlin - Spandau

Pate für: Kleinstadt
Fläche: 96,64 km² (in 16 Kacheln)
Größe: 820 / 1960 MB

CityObjects	284225
Buildings	48377
Addresses	51508
Thematic_Surfaces	235848
Surface_Geometries	860019
Appearances	131662
Surface_Data	441447
Texturparam	556373
CityObject_Genericattrrib	1670939
External_References	335605

Berliner Ring

Pate für: Innenstadt einer Großstadt
Fläche: 136 km² (in 35 Kacheln)
Größe: 1760 / 6710 MB

CityObjects	299370
Buildings	72585
Addresses	90526
Thematic_Surfaces	226765
Surface_Geometries	2247381
Appearances	211737
Surface_Data	995445
Texturparam	1795614
CityObject_Genericattrrib	1670939
External_References	375664

Berliner Stadtmodell

Pate für: komplette Großstadt
Fläche: 1596,4 km² (in 100 Kacheln)
Größe: 7460 / 21100 MB

CityObjects	2643887
Buildings	534357
Addresses	590746
Thematic_Surfaces	2109496
Surface_Geometries	9083266
Appearances	458721
Surface_Data	3935315
Texturparam	5187857
CityObject_Genericattrrib	9397597
External_References	3215838

Abb. 19b: Ausgewählte Datensätze im Überblick

Der Größe nach wurden die Datensätze wiederholt in eine leere *3DCityDB* importiert - jeweils einmal mit gleichzeitiger Indizierung der Daten und einmal ohne Indizes („deaktiviert"). Nachdem zuletzt das gesamte Stadtmodell mit Texturen importiert war, wurden daraus die Exporte gestartet. Mit dem Speicherbedarf in der Datenbank wurde ein weiterer Faktor systematisch dokumentiert. Dieser wurde mit der originären Größe der importierten Datei abgeglichen, die wiederum mit ihrem später exportierten Zwilling in den Byte-Zahlen verglichen wurde.

7.3.2. Datenbank- und System-Monitoring

Für das Monitoring der Datenbankprozesse und der Systemauslastung bieten sich interne Funktionen oder externe Programme wie z.B. *Nagios* oder *Munin* an. Bei letzteren ist die *Linux*-Herkunft von *PostgreSQL* an der mangelnden Unterstützung für die *Windows* Familie zu erkennen. Der Datenbank-Server wurde jedoch auf einem *Windows7*-Betriebssystem installiert. Für Windows-Nutzer wird das Analysieren mit dem *Windows Performance-Monitor* vorgeschlagen [www57], dem Pendant zu UNIX' *vmstat* und *iostat*. Um einen vergleichbaren Nenner gegenüber dem *Oracle*-DBMS zu haben, wurden alle Vorgänge parallel mit dem *Performance Monitor* überwacht. Die Auswahl an zu untersuchenden Parametern orientierte sich an [Hagander 2007].

Tab. 6: Überwachte Systemparameter [vgl. Edmead, Hinsberg 1998; Friedman, Pentakalos 2002]

Leistungsindikator	Erklärung
Processor (_Total):	
% Processor Time	Zeit, die der Prozessor für produktive Threads aufwendet
% User Time	Zeit, die für Benutzer bezogene Operation aufgewendet wird, i.d.R. Applikationen
% Priviliged Time	Zeit, die für auf den Systemkern bezogene Operation aufgewendet wird
Physical Disk (C:):	
% Disk Time	zeigt wie beschäftigt die Festplatte ist (ähnlich der Prozessor Time)
Avg Disk sec/Transfer	durchschnittliche Antwortzeit der Festplatte
Avg Disk Bytes/Transfer	durchschnittlich geschriebene und gelesene Bytes pro Festplattenantwort
Process (Postgres / Oracle):	
% Processor Time	Zeit, die der Prozessor für den Prozess aufwendet
% User Time	Zeit, die von der User Time für den Prozess aufgewendet wird
% Priviliged Time	Zeit, die von der Priviliged Time für den Prozess aufgewendet wird
Thread Count	Anzahl der Threads des Prozesses
Objects:	
% Processes	Anzahl an Prozessen im System
% Threads	Anzahl an Threads im System

Für *PostgreSQL* wurden darüber hinaus datenbankinterne Statistik-Tabellen und Log-Dateien inspiziert, um mögliche Flaschenhälse zu entdecken. Der php-Logfile-Analyzer *pgFouine* wurde nur stichprobenartig eingesetzt. Dieser scannt die Log-Dateien und dokumentiert langsame SQL-Abfragen in einem HTML-Bericht.

Kapitel 8

Ergebnisse des Performanz-Analyse

Aus den verschiedenen Testbedingungen generierte sich eine hohe Anzahl an Testdurchläufen, die mehrere Wochen dauern sollten. Für die CityGML-Importe waren jeweils für die *PostGIS*- wie für die *Oracle*-Version 84 Tests angesetzt, für die Exporte jeweils 44. Bei dieser Menge war es leider nicht möglich systematisch mehrere Durchläufe mit denselben Einstellungen durchzuführen, um eine statistische Signifikanz definieren zu können. Testwiederholungen wurden nur dann eingesetzt, wenn das Ergebnis gegenüber den bisherigen Zeiten des Datensatzes auffallend abwich.

Bei den später aufgeführten Testergebnissen wird sofort auffallen, dass keine Werte für eine *PostGIS*-Version mit getunten Einstellungen für *PostgreSQL* vorliegen. Dies liegt darin begründet, dass nach der Konfiguration des Datenbankservers keine einheitlichen Performanz-Steigerungen bei Im- und Exporten festzustellen waren. In den meisten getesteten Fällen dauerten die Prozesse sogar etwas länger. Die Einstellungen der Datei postgresql.conf wurde mehrere Mal zurückgesetzt und der Server mit leicht veränderten Parametern neu gestartet, ohne dass jedoch eine signifikante Veränderung gemessen werden konnte. Experten-Befragungen über Mailing-Listen konnten aus der Ferne nur vage Antworten liefern. Mit einem Benchmarking-Test ließen sich möglicherweise die optimalen Werte für die Software-Parameter tangieren. Dies war im zeitlichen Rahmen der Arbeit nicht durchführbar.

Eine weitere abhängige Komponente ist selbstverständlich die Hardware, mit der die Test durchgeführt wurden. Da der verwendete Server auch für andere Arbeiten des Instituts genutzt wird, konnte es zu Kollision von rechenintensiven Prozessen kommen. Um dies zu vermeiden wurden Tests, die länger als 5 Minuten benötigten, am Wochenende oder in der Nacht durchgeführt. Da der Zugriff und der Datentransfer über einen zweiten Rechner gesteuert wurde, ist die tatsächliche Signalstärke des Uni-internen Netzwerkes eine weitere Dunkelziffer. Die technischen Daten zur verwendeten Hardware sind in 1.4 verzeichnet.

In diesem Kapitel werden zuerst die Zeiten für den Import aufgeführt gefolgt von den Ergebnissen des CityGML-Exports. In 8.3 wird auf den Speicherbedarf in den DBMS eingegangen und in 8.4 auf die Beobachtungen des Monitorings.

8.1. Import von Datensätzen

Der Import von texturierten Daten dauerte bei einem einzigen Ausführungsthread deutlich länger und wurde deshalb nur für die kleinsten Datensätze durchgeführt. Die Tests wurden dennoch vorgenommen, um den Performanz-Vorteil des softwareseitigen Multithreadings messen zu können. Das texturierte Stadtmodell wurde ebenfalls aufgrund der langen Dauer nur mit den im *Importer/Exporter* standardmäßig eingestellten 8 Threads importiert. Sollten Diagrammkurven nicht vollständig sein, liegt dies an fehlenden Werten aufgrund von aufgetretenen Fehlern. Dies betraf jedoch nur Importe in eine *Oracle*-Datenbank. Zwei Arten von Fehlern wurden festgestellt:

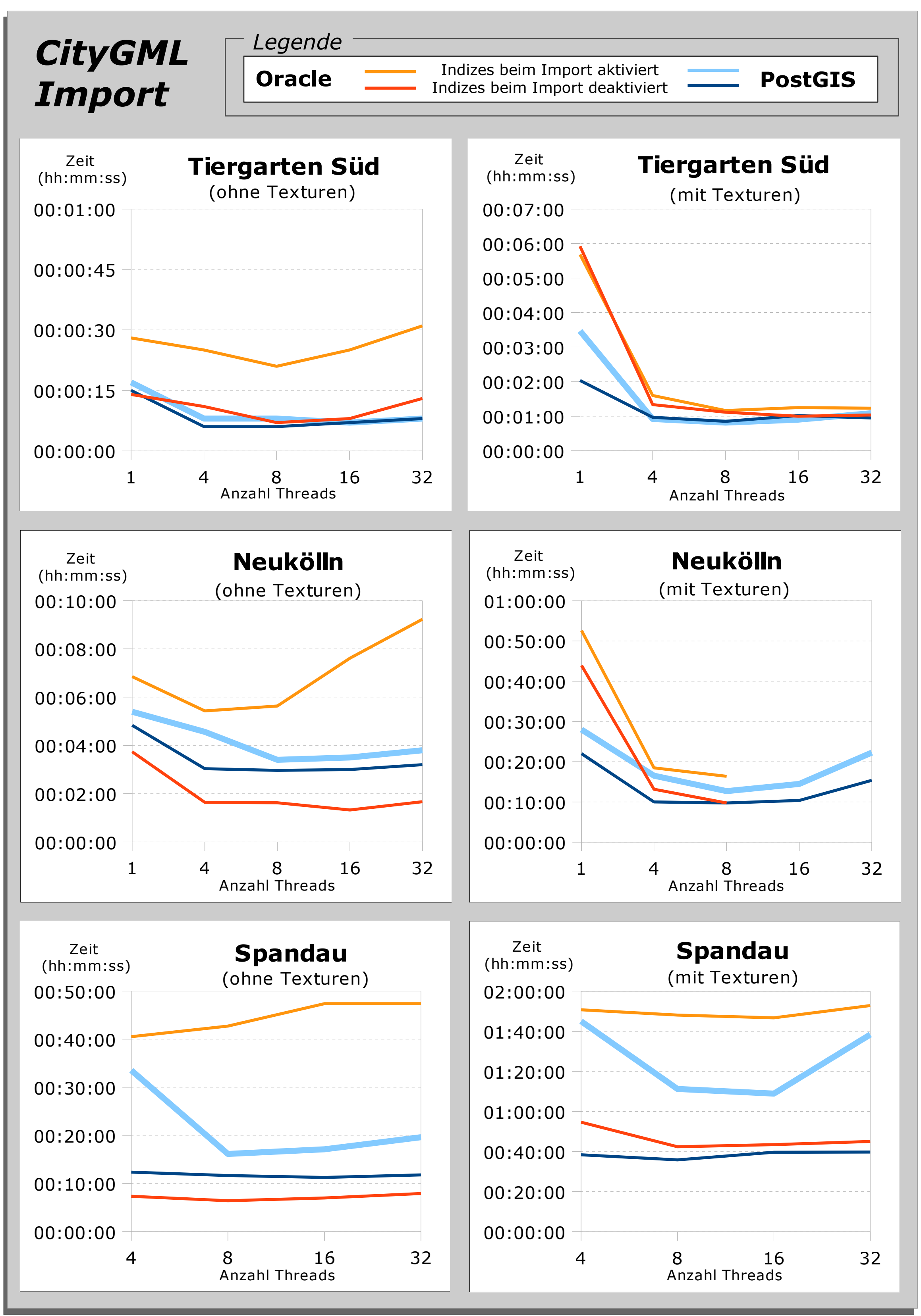
- Das Importieren von Texturen lies in der *Oracle*-Version bei zu hoher Anzahl an Texturdateien pro Datensatz (bzw. Kachel) den zugewiesenen Speicher (Java Heap Space) der Java Virtual Machine (JVM) überlaufen. In der Regel sollte das DBMS den für importierte Texturen benötigten Speicherplatz in der JVM wieder als frei ausweisen, sobald sich die Datei in der Datenbank befindet. Gegentests mit einer *Oracle 10g* Instanz verliefen bei den problematischen Datensätzen erfolgreich und belegen, dass es sich hierbei um einen Software-internen Bug von *Oracle 11g* im Zusammenhang mit dem ORDImage-Datentyp handeln muss.
- Bei Importen ab einer Threadanzahl von 16 konnte es passieren, dass für SQL-Abfragen keine Verbindungsobjekte (`Java.Connection`) mehr zur Verfügung standen. Der Prozess wurde entweder eingefroren oder fortgesetzt und der Import erfolgreich abgeschlossen.

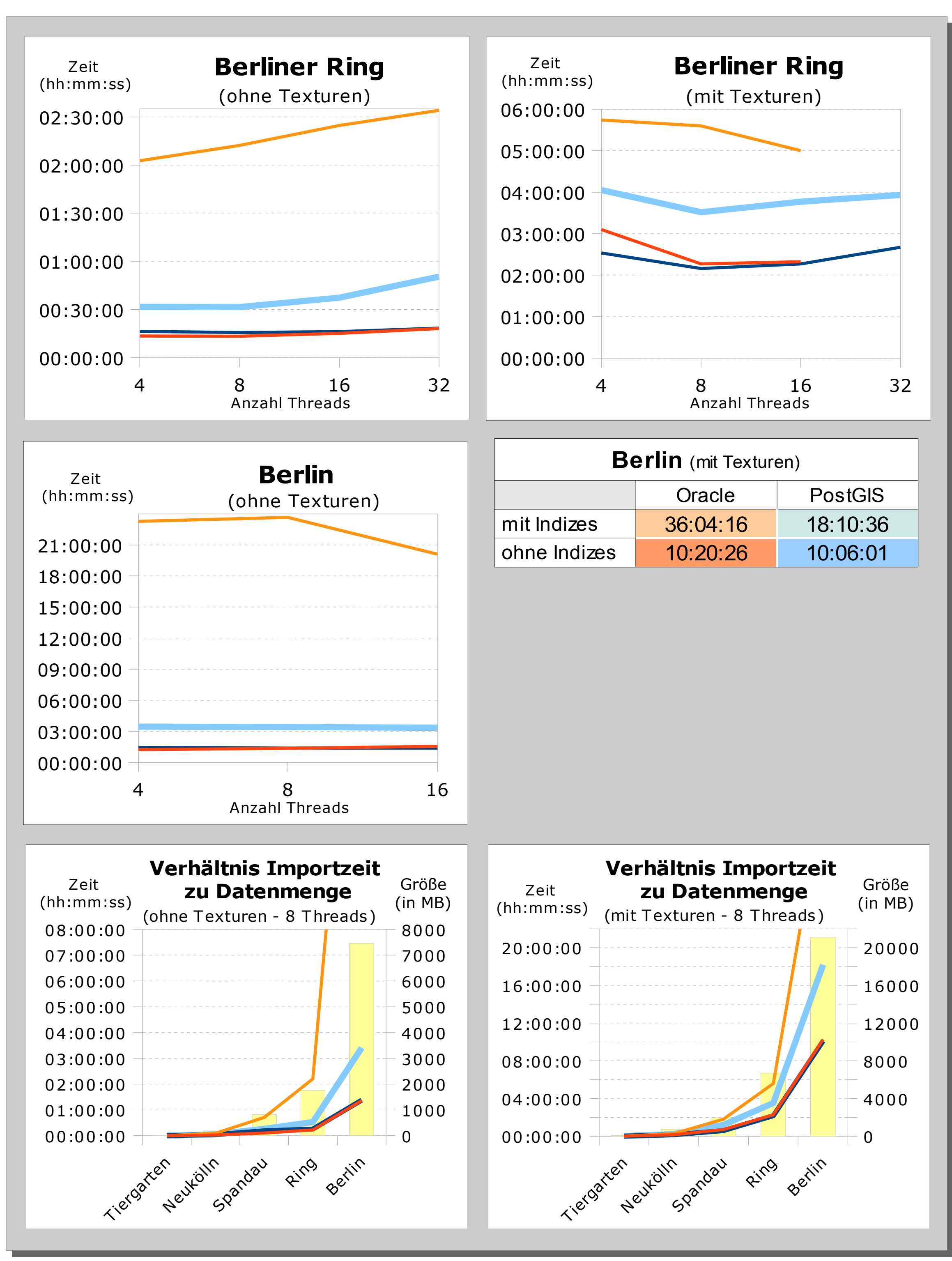
Im Falle von Neukölln und dem Berliner Ring verliefen die Importe für 4 und 8 Threads glücklicherweise erfolgreich und geben somit eine zeitliche Orientierung vor. Das gesamte Stadtmodell musste allerdings mit einer erheblich größeren Anzahl an Kacheln importiert werden, um erfolgreich in die Datenbank zu gelangen.

Folgende Beobachtung lassen sich zusammenfassend für die Importe konstatieren:

- Der Import dauerte deutlich länger, wenn die eingehenden Daten gleichzeitig für einen Index aufbereitet wurden. Da bei einem räumlichen Index jeder neue Eintrag eine aufwendige Umstrukturierung der hierarchische Baumstruktur des Indexes hervorrufen kann, war dieses Ergebnis zu erwarten. Bei nachträglicher Aktivierung ist die Lage aller Flächen bekannt und der Index wird in einem Prozess gemäß seines zu Grunde liegenden Algorithmus konstruiert.

Auf den folgenden zwei Seiten sind die Zeiten der Importtests in Diagrammen aufgeführt:





Berlin (mit Texturen)

	Oracle	PostGIS
mit Indizes	36:04:16	18:10:36
ohne Indizes	10:20:26	10:06:01

Abb. 20: Testergebnisse des Datenimports mit verschieden Datensätzen, variierender Anzahl paralleler Ausführungsthreads und jeweils aktivierten und deaktivierten Indizes

- Die zeitliche Auswirkung einer gleichzeitigen Indizierung der Daten ist in Tabelle 7 prozentual aufgelistet und ihre Entwicklung dem prozentualen Wachstum der Datensatzgröße in MB gegenüber gestellt. Diesbezüglich konnte jedoch kein proportionales Verhalten der Wachstumsraten festgestellt werden. Ansonsten ist in der Tabelle und auch in den Diagrammen gut zu erkennen, dass das zeitliche Gefälle für die *Oracle*-Version prägnanter ausfällt.

Tab. 7: Auswirkungen auf die Importzeit bei aktivierten gegenüber deaktivierten Indizes

Import	prozentualer Anstieg der Importzeit bei aktivierten gegenüber deaktivierten Indizes bei 8 Threads					Veränderung des Unterschiedes in % (erste Zeile = prozentuales Wachstum der Datensatzgröße)			
	Tiergart.	Neukölln	Spandau	Ring	Berlin	1088° / 847*	332° / 165*	115° / 242*	324° / 214*
PostGIS°	33	26	41	103	148	-23	60	150	44
Oracle°	200	293	1206	897	1622	46	312	-26	81
PostGIS*	4	31	100	63	80	669	220	-37	27
Oracle*	3	72	155	146	249	2468	116	-6	70

° ohne Texturen, * mit Texturen

- Insgesamt fiel das Verhältnis des Anstiegs der Dauer für einen Import bei wachsender Datenmenge mit der *PostGIS*-Version proportionaler aus. In der *Oracle*-Version verlängerte die Indexierung der Tabelleneinträge die Zeiten der Importe vor allem bei untexturierten Datensätzen in einigen Fällen um das doppelte bis dreifache gegenüber dem Mengenanstieg. Bei texturierten Datensätzen waren die Differenzen zwischen beiden DBMS hingegen gering. Von Neukölln zu Spandau und Ring zu Berlin war der zeitliche Anstieg jeweils doppelt so hoch wie das Wachstum der Dateigröße.

Tab. 8: Verhältnis Importzeit gegenüber ansteigender Datenmenge und -fläche

Import	Verhältnis Anstieg der Zeit im Vergleich zum Wachstum der Datensatzgröße um den jeweiligen Faktor (% / 100)			
	zu Neukölln	zu Spandau	zu Ring	zu Berlin
Größe+, Fläche+	11° / 8,5*, 18,5	3,5° / 1,5*, 4,5	1° / 2,5*, 1,5	3° / 2*, 12
PostGIS°^	39,5	4	1	5,5
Oracle°^	24,5	7	2	9,5
PostGIS°	42	3,5	0,5	4,5
Oracle°	18,5	1,5	3	5
PostGIS*^	23,5	4,5	2	4
Oracle*^	14	5,5	2	5,5
PostGIS*	18	3	2,5	3,5
Oracle*	8	3,5	2	3,5

° ohne Texturen, * mit Texturen, ^ Indizes aktiviert

Das Abweichen zu den anderen Datensatzsprüngen kann durch eine weitere Variable begründet werden: Der flächenhaften Ausdehnung eines Datensatzes. Je geringer der Flächenzuwachs desto proportionaler das Zeitverhalten des Imports im Bezug auf eine entsprechend größere Menge zu importierender Daten.

- Der zusätzliche Importe von Texturen verlängerte die Importzeit durch die höhere Datenmenge und Anzahl an Datenbankbefehlen (`INSERT`). Dieser zeitliche Unterschied fiel für die *Oracle*-Version bei gleichzeitiger Indexierung der Einträge teilweise geringer, bei deaktivierten Indizes teilweise höher aus als bei den *PostGIS*-Importen (siehe Tabelle 9).

Tab. 9: Auswirkungen auf die Importzeit bei Hinzunahme von Texturen

Import	Anstieg der Importzeit von texturierten gegenüber untexturierten Datensätzen verglichen mit dem relativen Anstieg der Datenmenge, jeweils in % / 100, bei 8 Threads				
	Tiergarten	Neukölln	Spandau	Berliner Ring	Berlin
Diff. MB	4	3	1,5	3	2
PostGIS^	5,5	3	3,5	5,5	4,5
Oracle^	4	2	1,5	1,5	0,5
PostGIS	7	2,5	2	7,5	6,5
Oracle	14,5	6	12	9,5	6,5

^ Indizes aktiviert

- Die Importe waren bei den meisten Testdurchläufen pro Datensatz bei 8 oder 16 Threads am schnellsten. Bis auf wenige Ausnahmen bei Importen mit aktivierten Indizes und abgesehen von den Tests mit einem einzigen Thread ergab eine Veränderung der Threadanzahl meist keine sprunghaften zeitlichen Abweichungen (siehe Diagramme).

8.2. Export von Datensätzen

Bei den Exporten kam es zu keinen Fehlern während der Arbeitsprozesse. Wie bei den Importen wurden auch Exporte von kleinen Datensätzen mit nur einem Thread durchgeführt, um den Unterschied in der Performanz beurteilen zu können. Eine starke Verkürzung der Exportdauer durch Multithreading ergab sich aber nur bei der *Oracle*-Version.

Folgende Beobachtung können aus den Exportergebnissen abgeleitet werden:

- Die Exporte liefen bei *Oracle* häufig schneller durch. Besonders bei untexturierten Exporten (d.h. auch ohne Appearance-Einträge im CityGML-Dokument) ergab sich ein deutlicher Unterschied in der Ausführungszeit (siehe Diagramme). Es wurde beobachtet, dass bei der *PostGIS*-Version relativ viel Zeit vergehen konnte (5 Minuten für ganz Berlin), bevor überhaupt Objekte exportiert wurden.

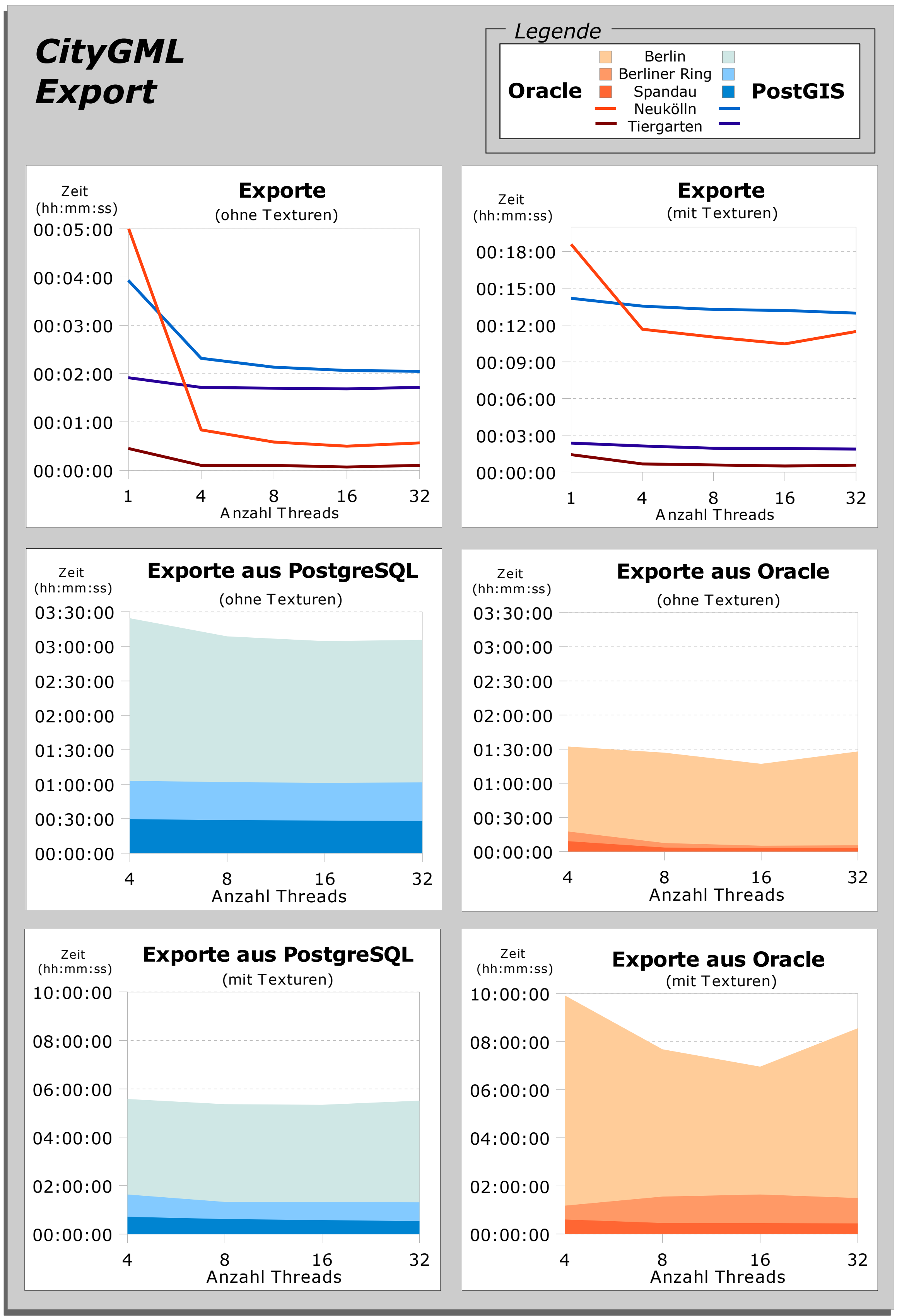


Abb. 21: Testergebnisse des Datenexports bei variierender Anzahl paralleler Ausführungsthreads

Bei der *Oracle*-Version starteten die Zähler des *Importer/Exporter* dagegen im Verhältnis ohne große Verzögerung. Eine Analyse der *PostgreSQL*-Log-Dateien ergab, dass die räumliche Filterung mit der Bounding Box zu Beginn des Exportprozesses sehr viel Zeit benötigte, die restlichen Schritte jedoch ähnlich schnell verliefen wie beim *Oracle*-Export.

- Die texturierten Exporte bewegten sich zeitlich bei beiden Versionen auf einem ähnlichen Level. Die *Oracle*-Version benötigte ab dem Berliner-Ring-Modell mehr Zeit, was gleichzeitig einem größeren prozentualen Unterschied zu untexturierten Exporten bedeutet. (siehe Tabelle 10).

Tab. 10: Auswirkungen auf die Zeit beim zusätzlichen Schreiben von Texturdateien

Export	Anstieg der Exportzeit von texturierten gegenüber untexturierten Datensätzen verglichen mit dem relativen Anstieg der Datenmenge, jeweils in % / 100, bei 8 Threads				
	Tiergarten	Neukölln	Spandau	Berliner Ring	Berlin
Dif. Größe	4,5	4	2	3,5	2,5
PostGIS	0,1	5,5	0,5	0,5	1
Oracle	3,5	37	7,5	12	4,5

- Im Verhältnis zwischen Dauer und exportierter Datenmenge fällt es schwer Regelmäßigkeiten zu finden. Anstiegsraten bei der Exportzeit von untexturierten Stadtmodellausschnitten sind mal höher als bei texturierten Daten, mal niedriger. Ein einheitliche Korrelation zur flächenhaften Veränderung der Datensätze liegt demnach nicht vor.

Tab. 11: Verhältnis Exportzeit gegenüber ansteigender Datenmenge und -fläche

Export	Verhältnis Anstieg der Zeit im Vergleich zum Wachstum der Datensatzgröße um den jeweiligen Faktor (% / 100) bei 8 Threads			
	zu Neukölln	zu Spandau	zu Ring	zu Berlin
Größe+, Fläche+	9,5° / 8*, 18	3,5° / 1,5*, 4,5	1° / 2,5*, 1,5	4° / 2,5*, 12
PostGIS°	0,5	12,5	1	5,5
Oracle°	3,5	8,5	1	11
PostGIS*	7,5	2	1	3
Oracle*	34	1	2,5	4

° ohne Texturen, * mit Texturen

- Die Exporte ohne Texturen waren für einen jeweiligen Datensatz bei beiden Versionen mit 16 Threads am schnellsten. Mit Texturen wurde die Bestzeit für alle Datensätze mit Ausnahme vom gesamten Stadtmodell bei 32 Threads erreicht. Wie bei den Importen führte das Verändern des Thread-Parameters zu keinen markanten zeitlichen Sprüngen (siehe Diagramme).

8.3. Speicherbedarf

Der Speicherbedarf wurde in Megabyte gemessen. Die Größe einer leeren *3DCityDB* betrug in *Oracle* 34 MB und in *PostgreSQL* 14 MB, inklusive des eigentlichen Speicherplatzes des Arbeitsbereiches (*Oracle*) bzw. der Datenbank (*PostgreSQL* inkl. *PostGIS*). Folgende Werte wurden dokumentiert:

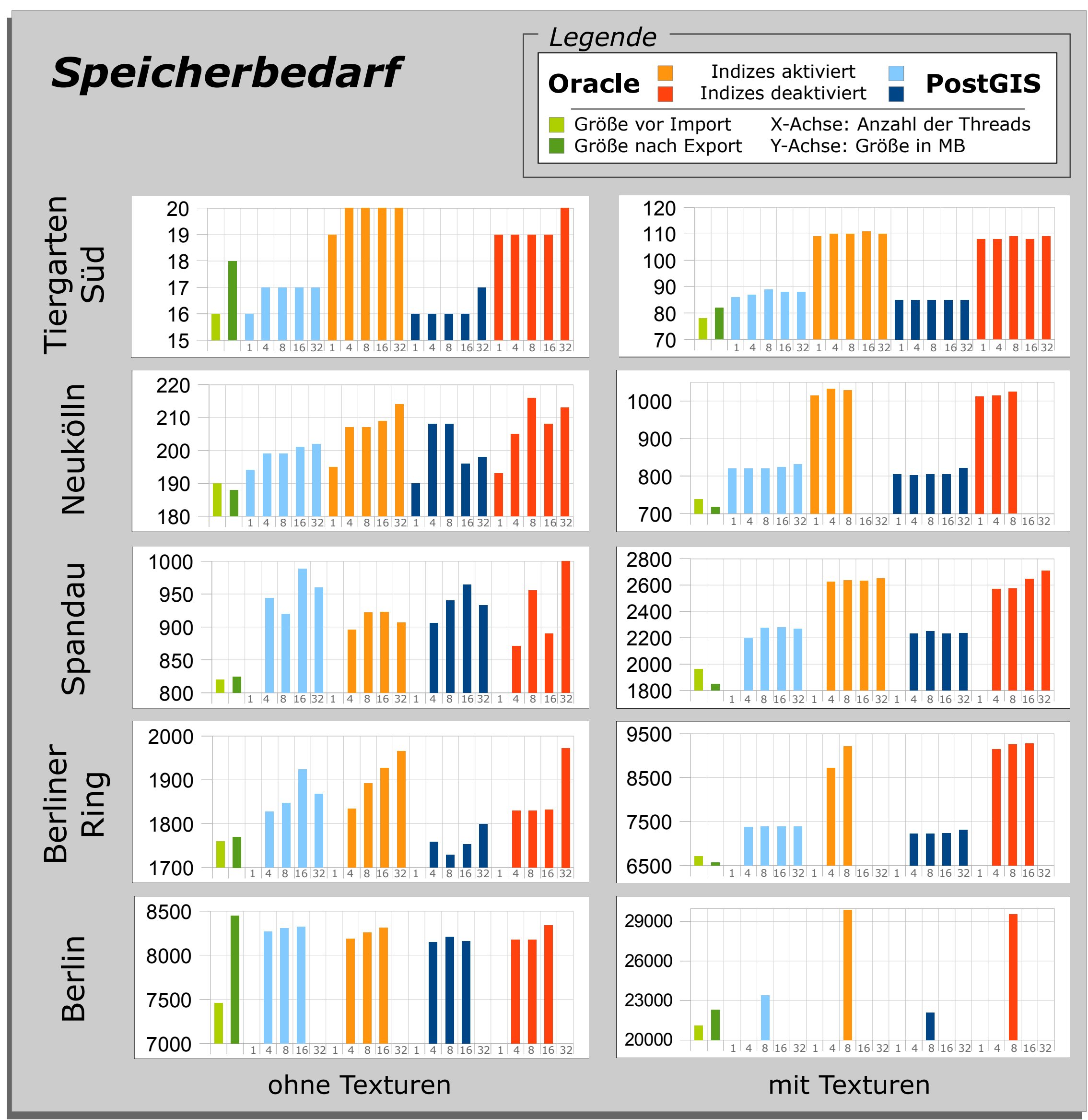


Abb. 22: Speicherbedarf der Datensätze in Oracle und PostgreSQL

Der Zustand „Indizes deaktiviert" bezieht sich nur auf den Importprozess. Da ein Index ebenfalls Speicher benötigt, wurden die Indizes nach dem Import mit dem *Importer/Exporter* erstellt, damit die Werte mit dem Zustand „Indizes aktiviert" vergleichbar blieben.

Folgende Beobachtung ergeben sich aus den Ergebnissen der Speichermengen:

- Während sich der Speicherbedarf bei untexturierten Datensätze gemessen an der MB-Zahl in beiden DBMS die Waage hält, ist bei zusätzlicher Speicherung von Texturen ein Unterschied zwischen *Oracle* und *PostgreSQL/PostGIS* von durchschnittlich 25% feststellbar.
- Bei höherer Anzahl an Threads stieg häufig auch der Speicherbedarf im DBMS.
- Wenn die Indizes erst nachträglich aktiviert werden, fällt der Speicherbedarf fast jedes Mal etwas geringer aus.
- Die Größe der exportierten Dateien entsprach in keinem Fall der Größe der Ausgangsdateien. Die Verkleinerung bei texturierten Daten kann damit erklärt werden, dass während des Importes kleinste Texturen, die nur eine Farbe beinhalteten, als 0 byte Textur angesehen und beim Import übersprungen wurden.

8.4. Rechnerauslastung

Die Prozessorauslastung bei untexturierten Importen betrug bei der *PostGIS*-Version des *Importer/Exporter* durchschnittlich ca. 50% bei deaktivierten Indizes, bei aktivierten ca. 70%. Bei *Oracle* war das Verhältnis 20% zu 90%. Wenn die Texturen mit importiert wurden betrug die Auslastung ca. 45% und 97% gegen 30% und 60%. Bei beiden Version konnte bei Verdoppelung der Threads ein Anstieg der Auslastung um etwa 10% beobachtet werden. Bei den Export-Tests (*PostGIS* und *Oracle*) blieb die Auslastung bei variierender Anzahl an Threads und Größe der Datensätze meist konstant um 35% (+-10%). Durch den höheren Aufwand des Systemkerns bei vielen Ausführungsthreads stieg der Leistungsindikator Privileged Time gegenüber der User Time (siehe 7.3.2).

Auf allen getesteten Rechnern mit Windows Betriebssystem belegt der *Oracle*-Prozess mit Abstand den meisten Platz im Arbeitsspeicher (mehrere 100 MB). Ein *PostgreSQL*-Prozess belegt im Vergleich etwa 99% weniger. *PostgreSQL* bildet hingegen je nach Workload mehrere Prozesse aus, die eine bestimmte Anzahl an Threads besitzen (der erste Prozess 4, alle weiteren 3). Bei Im- und Exporten mit 32 Threads erhöhte sich die Threadanzahl im ersten Prozess um 1 und die Anzahl der Prozesse belief sich teilweise auf bis zu 70 (38 bei eingestellten 16 Ausführungsthreads, 22 bei 8 Threads etc.). Für *Oracle* wird bei steigendem Aufwand die Anzahl der Threads des einzigen Prozesses erhöht, z.B. bis zu 150 bei eingestellten 32 Ausführungsthreads.

Bei den Leistungsindikatoren für das Festplattenlaufwerk C:\, auf dem die Daten des *Oracle* und *PostgreSQL* DBMS gespeichert waren, ergab sich kein einheitliches Bild für eine jeweilige *Importer/Exporter*-Version. Allerdings waren die Werte bei den *Oracle*-Tests gegenüber *PostgreSQL/PostGIS* häufig 10, teilweise bis 100 Mal höher. Auch bei Stichproben mit einem getunten *PostgreSQL* DMBS blieb der Unterschied ähnlich groß.

Kapitel 9

Bewertung der Ergebnisse

Die Portierung kann nicht nur aufgrund einer funktionierenden Version des *Importer/ Exporter*, sondern vor allem wegen der ähnlichen Ergebnisse beim Vergleich mit der bisherigen *Oracle*-Version als Erfolg gewertet werden. In diesem letzten Kapitel werden die Ergebnisse in 9.1. genauer interpretiert und Charakteristika für beide DBMS identifiziert, mit denen sich endgültige Aussagen zur Performanz relativieren und somit auch präzisieren lassen. In 9.2. wird die geleistete Arbeit zur Portierung reflektiert, um den Aufwand für zukünftige Entwicklungen bemessen zu können. 9.3. komplettiert die Beantwortung der Ausgangsfragestellungen durch konzeptionelle Vorschläge für bisher nicht direkt übersetzbare Teile des Software-Paketes bei gleichzeitiger Diskussion, wie sich das Fehlen jener Komponenten auf den praktischen Betrieb der *3DCityDB* auswirkt.

9.1. Interpretation der Ergebnisse

Die vorliegenden Ergebnisse liefern interessante Erkenntnisse, reichen jedoch nicht aus, um mögliche Hypothesen zu verifizieren oder zu widerlegen. Dafür waren die Testbedingungen für beide DBMS zu unterschiedlich, was die folgende Unterpunkte detailliert analysieren werden.

Außerdem konnte für *PostgreSQL* keine optimale Konfiguration des Datenbankservers an das Betriebssystem erreicht werden. *Oracle 11g* hat dadurch den Vorteil, die Rechnerressourcen potentiell besser ausnutzen zu können und eine höhere Leistung zu erzielen, was u.a. das Monitoring des Festplattentransfers offenbart. Für eine genauere Analyse der etwa gleich hohen Prozessorauslastung hätte dezidiert beobachtet werden müssen, in wie weit die jeweilige Datenbank-Software ihre Prozesse nebenläufig verarbeitet. Die Auslastung steht bei der *PostGIS*-Version im Kontrast zu den niedrigen Konfigurationen von *PostgreSQL*.

9.1.1. Welches DBMS war schneller?

Die portierte Version erreichte trotz minimaler Standardeinstellungen von *PostgreSQL* gleichwertige Zeiten. Das Open-Source-DBMS kann also durchaus als eine schnelle Software bezeichnet werden. Spannend bleibt die Frage, wie schnell die *Oracle*-Version jeweils gewesen wäre, würde der BLOB-Datentyp für die Speicherung von Texturen eingesetzt werden? Die wachsende zeitliche Differenz zwischen den Versionen bei Exporten von texturierten Daten zeigt, dass das Schreiben von Texturdateien aus ORDImage-Objekten länger dauert als bei den binären Arrays von *PostgreSQL*. Das liegt sehr wahrscheinlich an der Größe der *Oracle*-spezifischen Objekte (siehe Speicherbedarf). Die Werte in Tabelle 9 belegen, dass auch der Import durch ORDImages stärker gebremst wird, als dies möglicherweise bei BLOBs der Fall wäre. Bei deaktivierten Indizes beläuft sich der Anstieg der Zeit zwischen Importen von texturierten und untexturierten Datensätzen durchschnittlich auf den Faktor 10,7 bei *Oracle* und 5,4 bei *PostgreSQL*. In Abbildung 23 ist dieser Vergleich zusammen mit den zeitlichen Unterschieden durch die Indexierung der Daten nochmals dargestellt.

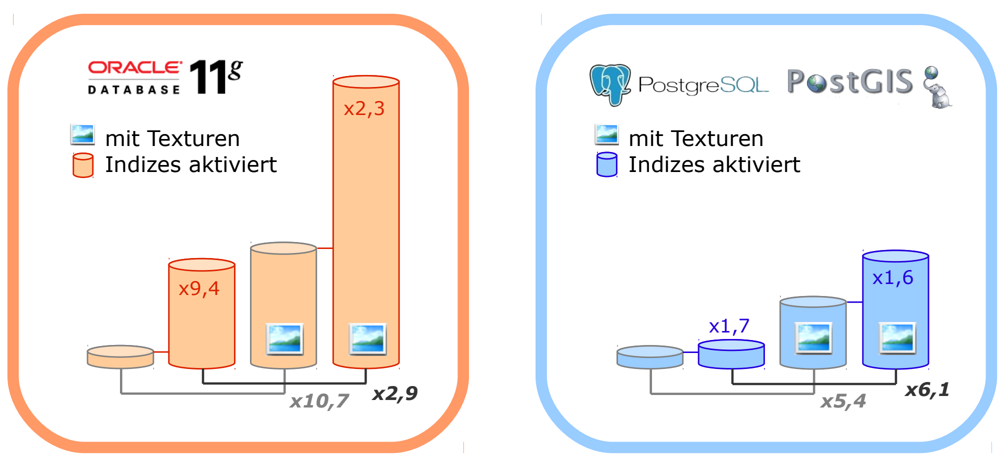


Abb. 23: Relative zeitliche Verlängerung bei Importen bei Hinzunahme von Texturen und bei gleichzeitiger Indizierung der Daten

Es fällt auf, dass die Hinzunahme von Texturen bei *PostgreSQL* für die indizierende und die nicht indizierende Import-Variante einen ähnlichen additiven Einfluss auf die Zeit hat, während bei *Oracle* die zusätzlichen Datenbank-Einträge der Texturen für den Index weniger ins Gewicht fallen. In der Regel dauert das Erstellen eines Indexes für räumliche Daten (R-Baum) länger als bei einem Index für sortierbare Daten (B-Baum), da die Restrukturierung des Indexbaumes bei hinzukommenden Werten viel aufwendiger sein kann [vgl. Brinkhoff 2008: 161ff.]. Die Spalten aus den Tabellen für die Texturen (`Surface_Data` und `Appearance`) sind mit einem B-Baum-Index indiziert. Es ist daher anzunehmen, dass zusätzliche zu indizierende Einträge in diese Tabellen den Import nicht im selben Verhältnis verlangsamen können, wie es jene Indizes tun, die beim untexturierten Import zum Einsatz kommen (z.B. für die Geometrien der Gebäude).

Der GiST-Index von *PostgreSQL* scheint wegen gleicher Anstiegsraten genau so schnell konstruiert zu werden wie der B-Baum-Index. Die bei dem Datenbank-Export für den Bounding Box Filter zum Einsatz kommende *PostGIS*-Funktion `ST_Relate` sollte den erstellten GiST Index verwenden. Wie in 8.2 erläutert, dauert die Abfrage deutlich länger als das *Oracle*-Pendant. Die Überprüfung der Statistik-Tabelle `pg_stat_user_indexes` ergab, das die in Frage kommenden räumlichen Indizes kaum vom SQL-Planer des freien DBMS verwendet wurden (Spalten: `idx_scan`, `idx_tup_read`, `idx_tup_fetch`). Auch das programmierte VACUUM-Skript konnte zu keiner Erhöhung dieser Werte beitragen. Dies lässt Zweifel an der Funktionalität des Indexes entstehen. Ein weiteres Fein-Tuning könnte die Zeiten für den Export möglicherweise stark verkürzen, aber ggf. auch die Importe mit aktivierten Indizes verlängern.

Letzteres wäre für den praktischen Einsatz der Software unrelevant, weil das Deaktivieren der räumlichen Indizes vor dem Import ohnehin zu empfehlen ist, um das Einladen der Daten zu beschleunigen. Für das gesamte Stadtmodell dauerte die nachträgliche Indizierung bei der *PostGIS*-Version ohne Texturen 10 Minuten und mit Texturen 12. Bei *Oracle* waren es 22 und 29 Minuten. Dies zeigt nochmals den Unterschied zwischen den Indizes auf, der hauptsächlich für die längeren Importzeiten der *Oracle*-Version verantwortlich ist. Entscheidend ist jedoch ein funktionierender Index, der schnelle Abfragen ermöglicht. Hier leistete *Oracle Spatial* bessere Arbeit als *PostgreSQL/PostGIS*.

9.1.2. Welches DBMS skalierte besser?

Anhand der Werte aus den Tabellen 7 bis 11 lässt sich zusammenfassen, dass sich der zeitliche Anstieg im Bezug auf die Datensatzgröße bei der *PostGIS*-Version gleichmäßiger verhalten hat. Beide DBMS konnten eine jeweilige Verdoppelung der Ausführungsthreads ohne auffallende Performanz-Einbußen verarbeiten (bis auf die in 8.1 erwähnten Fehler bei *Oracle*). Für präzisere Aussagen wäre allerdings eine systematische Auswahl an Datensätzen erforderlich gewesen. Ein regelmäßige Abstufung der Dateigröße bei gleichbleibender Fläche bzw. eine wachsende Ausdehnung bei gleicher Dateigröße wäre für zukünftige Untersuchungen denkbar. Die Dateigröße ließe sich darüber hinaus nach der Menge an Gebäuden gliedern, z.B. um zu ermitteln, wie lange der Import von vielen LOD 1 gegenüber wenigen LOD 2 Gebäuden dauern würde, die in der Summe den gleichen Speicherplatz belegen. Für eine daraus resultierende Ergebnismatrix würden dann auch Varianzwerte für die Zeitkurven bei den gewählten Thread-Stufen sinnvoll sein.

Was die Robustheit der DBMS betrifft, so fiel *Oracle* durch den Bug um ORDImages negativ auf. Bei einer Open-Source-Software genügt es meist, einen solchen Fehler einer entsprechenden Mailing-List mitzuteilen, damit sich die Entwickler des Projektes dem Problem annehmen. Bei einem proprietären Produkt wie *Oracle 11g* müsste entweder ein umfassender Bug-Report verfasst oder der Fehler umgangen werden.

Des weiteren treten bei *Oracle* Verklemmungen („Deadlock") von Threads beim Erstellen von STRUCT-Objekten mit der Methode JGeometry.store auf, d.h. zwei Ausführungsstränge warten darauf, dass der jeweils andere Strang die Methode ausführt. In diesem Fall muss der komplette Import von Geometrien über eine separate Warteschlange synchronisiert werden. Die entsprechenden Klassenmethoden des *PostGIS* JDBC-Treibers waren bei allen Tests „threadsafe", d.h. es traten keine Verklemmungen auf. Ein daraus resultierender Performanz-Vorteil gegenüber der *Oracle*-Version konnte jedoch nicht festgestellt werden (siehe Importe ohne Texturen und aktivierten Indizes). Die höhere Transferrate vom *Oracle*-Prozess mit der Festplatte könnte in diesem Falle ausgleichend wirken.

Verklemmungen sind bei einer auf Multithreading ausgelegten Datenbank-Applikation nichts Unübliches. Für die *PostGIS*-Version mussten in einigen Java Klassen das Festschreiben von Änderungen in der Datenbank („commit") in der Taktung erhöht werden, was die Performanz in der Regel drosselt. Dies betraf zum Einen das Auflösen von XLink-Referenzen mit einhergehenden UPDATE-Befehlen an die Datenbank. Ein anderes tückischeres Problem, weil ohne Fehlermeldung in Java („Exception"), betraf die Server-Prozesse von *PostgreSQL*. Wie in 8.4 beschrieben, bildet *PostgreSQL* mehrere Rechnerprozesse aus, anstatt wie *Oracle* die Threadanzahl eines Prozesses zu erhöhen. Jeder Prozess wird als eine eigenständige Verbindung zu Datenbank, auch „Session" genannt, angesehen. Änderungen aus anderen Sessions, die nicht direkt festgeschrieben werden, können eine Abfrage in einer bestimmten Session unendlich lange laufen lassen. Über die System-Tabelle pg_stat_activity kann der Benutzer diese fehlerhaften Transaktionen an dem Status <IDLE> in transaction identifizieren. Das Problem multipler Sessions hat *Oracle* nicht.

9.2. Reflexion zu geleisteten und zukünftigen Portierungen

Die manuelle Portierung der *3DCityDB* und des *Importer/Exporter* war in erster Linie eine Fleißarbeit. Das Vertiefen der Programmierkenntnisse in SQL und Java hätte auch mit der Fokussierung auf die ausschließliche Verarbeitung von Gebäuden erreicht werden können, weil die Funktionsweise für die anderen Klassen des Datenmodells nahezu identisch ist. Eben deshalb verursachte die vollständige Konvertierung von Schema und CityGML Import/Export nur einen geringen Mehraufwand und sorgte simultan für ein umfassenderes Verständnis des Datenmodells und wachsende Routine im Umgang mit beiden DBMS. Die Entscheidung alle thematischen Module umzusetzen fiel auch im Hinblick auf eine zukünftigen Veröffentlichung der Software.

Der Anspruch bei der Übersetzung der Unterschiede in den SQL-Skripten war gering. Konverter würden sich für die zukünftige parallele Entwicklung beider Versionen anbieten, ohne dass der Umfang an mögliche Nachbesserungen unüberblickbar wäre. Aufgrund der eingesetzten Spezifika von *PostGIS 2.0*, sollte der Konverter diese Version unterstützen. Erweiterungen oder die komplette Konvertierung des relationalen Schemas auf ein ander-

es DBMS (z.B. *SQL Server*) würde nur wenige Tage in Anspruch nehmen. Anders könnte es für die Portierung von SQL-Skripten aussehen, die eine Operation auf die Datenbank ausführen. Zum Beispiel verfügt *Oracle* über mehr Fähigkeiten als *PostgreSQL* mit den Funktionen von psql, so dass ggf. auf eine prozedurale Sprache ausgewichen werden müsste. Das Programmieren von Stored Procedures bietet durch seine Kontrollstrukturen mehr Möglichkeiten beim Automatisieren von Prozessen, ist aber entsprechend aufwendiger in der Programmierung. Die Arbeit hat gezeigt, dass die Konvertierung von *Oracle* PL/SQL-Skripten nach PL/pgSQL durch die Verwendung Oracle-spezifischer objekt-orientierter Merkmale und abweichenden Systemtabellen zu *PostgreSQL* weit weniger trivial ausfiel. Mit entsprechender Praxis gleicht das Übersetzen aber dem Dolmetschen bei linguistischen Sprachen und der Portierungsaufwand sinkt wesentlich. Das Anpassen der restlichen PL/SQL-Skripte wäre ein Angelegenheit von einigen Tagen.

Das Umprogrammieren der betroffenen Java-Klassen erforderte aufgrund der großen Abweichungen zur *Oracle*-Version ein gewisses Maß an Kreativität. Erschwert wurde die Umsetzung durch die bereits erwähnte mangelnde Dokumentation zum JDBC-Zugriffs auf *PostGIS*. Diese Arbeit vermittelt deshalb auch eine generelle Orientierung für das Programmieren einer Java-Applikation für *PostGIS*. Sobald ein funktionierendes Konzept entwickelt war, konnte es schnell in allen betroffenen Klassen implementiert werden. Weitere Portierung, z.B. vom *KML-Exporter* können darauf aufbauen. Bei abweichenden Algorithmen oder Abfragen an die Datenbank kann das Umprogrammieren eine zeitintensive Arbeit bleiben. Häufig sind es jedoch viel kleinere Details, die den Entwicklungsprozess über Wochen in die Länge ziehen können. Seien es fehlerhafte Stellen im Quellcode, die bei vorherigen Testläufen nicht angesprochen wurden, oder plötzlich auftretende Verklemmungen von Threads bei Datenbankoperationen mit bestimmten Datensätzen.

Das Fein-Tuning aller Software-Bestandteile ist in den vorangegangen Abschätzungen nicht mit einbezogen. Die SQL-Skripte und der Java-Code wurden größtenteils direkt konvertiert, wobei nicht zwangsläufig die beste Lösung für *PostGIS* entstanden sein muss. Da die Software quelloffen ist, besteht aber die Möglichkeit zu einem Austausch mit Experten in einer öffentlichen Diskussion.

9.3. Alternativen für nicht direkt portierbare Komponenten

Die erfolgreiche Teilumsetzung hat bewiesen, dass es möglich ist die *3DCityDB* und den dazugehörigen *Importer/Exporter* nach *PostgreSQL/PostGIS* zu portieren. Anhand der in Kapitel 6 vorgestellten Methoden kann auch für fast alle Komponenten (auch *Matching/Merging*-Plugin und *KML-Exporter*) bestätigt werden, dass die Portierung zwar abweichende Konzepte erfordert, aber inhaltlich nahezu verlustfrei möglich ist.

Ein offener Punkt, der auch nach einer möglichen Veröffentlichen der *PostGIS*-Version weiterhin existieren wird, ist die fehlende Historienverwaltung der *3DCityDB*. *PostgreSQL* bietet keine Versionsverwaltung einer Datenbank an. Alternativ müsste entweder eine

eigene Logik innerhalb des DBMS programmiert werden, auf welche dann die *Planning Manager* Skripte zugreifen, oder die Inhalte des DBMS würden mit einer externen Versionierungssoftware verwaltet werden (*Git*, *TortoiseSVN*, *Apache Subversion*). In jeden Fall müsste eine hierarchische Versionierung wie mit dem *Oracle Workspace Manager* ermöglicht werden. Eine Datenbankseitige Lösung für *PostGIS* existiert z.B. mit *pgVersion* [www58]. *pgVersion* realisiert in seiner jetzigen Version aber nur eine lineare Versionierung für das konkurrenzierende Editieren von Geometrien einzelner Tabellen und nicht von Inhalten eines gesamten Datenbank-Schemas.

Für eine erste Veröffentlichung der *PostGIS*-Version ist eine fehlende Historienverwaltung nicht zwingend, da es in erster Linie um die Bereitstellung der Stärken der *3DCityDB* auf Basis eines komplett freien Software-Paketes geht. Langfristig werden sich Stadtmodelle ohne die Möglichkeit einer zeitlichen Datenarchivierung kaum Speicherplatz-schonend verwalten lassen. Am Beispiel des Projektes „Amtliches Berliner Stadtmodell“ kann festgemacht werden, dass diese Funktionalität von einem DBMS gefordert wird und *PostGIS* somit nicht in Frage käme. Bei anderen bereits existierenden Stadtmodellen liegt die Beeinträchtigung durch fehlende Versionierung bei einer möglichen Verwendung der *3DCityDB* in *PostGIS* im Ermessen des Auftraggebers. Für neue Projekte würde sich die *PostGIS*-Version bei dem Aufbau einer Datenhaltungskomponente für Stadtmodelle anbieten. Gerade Stadtverwaltungen von kleineren Kommunen, deren Etat keine Lizenzierung eines *Oracle* Servers zulässt, erhalten eine kostenfreie Alternative und können ihre Ressourcen in die Entwicklung von Werkzeugen investieren, die speziell auf ihre Bedürfnisse abzielen. So könnte zwangsläufig ein gleichwertiger Ersatz zum Historienmanagement geschaffen werden.

Kapitel 10

Fazit

Während der Monate in denen diese Arbeit verfasst wurde, hat der Entwicklungsstand der *PostGIS*-Portierung unlängst die gesteckten Ziele einer Testanwendung überschritten und ist zu einer offiziell veröffentlichten Version der *3DCityDB* und des *Importer/Exporter* gereift. Ein Installations-Setup kann von der Projekt-Homepage [www54] herunter-geladen werden. Über die Teilumsetzung mit dem relationalen Schema und dem CityGML *Importer/Exporter* hinaus wurde auch das *KML-Exporter*-Modul konvertiert und erfolgreich mit verschiedenen Datensätzen getestet. Durch die Übersetzung der restlichen PL/SQL-Skripte kann auch das *Matching/Merging* Plugin für den *Importer/Exporter* zur Verfügung gestellt werden. Der Aufwand der Vervollständigung der Portierung entsprach den getätigten Abwägungen, obwohl einige bis dato unbekannte Algorithmen zu übersetzen waren (siehe Anhänge). Einzig das Historienmanagement musste vorerst ausgeklammert werden.

Wie zuletzt deutlich wurde, könnte dieses offengeblieben Feature zukünftig mit alternativen Lösungen besetzt werden, die mit dem Aufbau von neuen Datenbanken für 3D-Stadtmodelle projektspezifisch entwickelt würden. Für das von der EU geförderte Projekt *i-SCOPE* [www59] sollen in naher Zukunft etwa 15 CityGML-Pilotprojekte verteilt über Europa entstehen, die auf der Grundlage der entwickelten *PostGIS*-Version aufbauen werden. Damit kann sich die Software nicht nur in einem sehr interessanten Anwendungsfall beweisen, gleichzeitig trägt sie, wie in der Einleitung prognostiziert, zu einer Verbreitung von CityGML und auch der *3DCityDB* bei.

Rückwirkend betrachtet ist die Umsetzung von *Oracle Spatial* nach *PostGIS* methodisch leichter zu bewältigen gewesen, als es anfangs abzusehen war, und konnte auch nur deshalb noch während der Masterarbeit fertig gestellt werden. Es sind die international festgelegten Standards wie SQL (und dessen Erweiterungen SQL/PSM, SQL/MM Spatial) und Simple Features, die von allen Herstellern räumlicher RDBMS übernommen werden, und somit garantieren, dass sich die grundlegenden Konzepte auch bei teils unterschied-

licher Auslegung nie zu weit von einander entfernen. Dadurch können schneller Lösungsansätze bei größeren Unterschieden zwischen *Oracle Spatial* und *PostGIS* gefunden werden.

Hinzu kam, dass kaum Anpassungen an die auf Multithreading ausgelegte Architektur des *Importer/Exporter* notwendig waren und sich *PostgreSQL* seit den ersten Tests als eine robuste Software erwies. Das Verändern von Testparametern führte meist zu einer gleichmäßig variierenden Ausprägungen der Ergebnisse. Die Performanz-Analyse hat einige Charakteristika beider DBMS aufgedeckt, aber auch neue Fragen hervorgerufen, die zukünftige Entwicklungen berücksichtigen müssen bzw. systematischere Testverfahren untersuchen können. Dies betrifft vor allem die Indizierung räumlicher Daten in *PostgreSQL* und den Vergleich von ORDImage zu BLOB in *Oracle*.

Die *PostGIS*-Unterstützung ist von nun an ein fester Bestandteil bei der Weiterentwicklung der *3DCityDB* und des *Importer/Exporter*. Das Zusammenführen beider Versionen ist das langfristige Ziel und wird neben einer Anpassungen auf CityGML 2.0.0 den Aufbau der Software nachhaltig verändern. Der Ausbau der 3D-Unterstützung von *PostGIS* sowie das Herstellen von Schnittstellen für das offline (GIS) und online (WFS, W3DS, WVS) Management von 3D-Stadtmodellen sind spannende zukünftige Themen. Der Open-Source-Gedanke wird dabei ein dauerhafter Begleiter bleiben und hat nicht zuletzt in dieser Masterarbeit sein Potential bewiesen.

Verzeichnisse

In der digitalen Fassung der Masterarbeit sind die Einträge in den Verzeichnissen mit den Textstellen verlinkt und umgekehrt. Die entsprechende Seitenzahl befindet sich am Ende des Verzeichniseintrages. Alle Internetquellen wurden das zuletzt am 16. September 2012 aufgerufen.

Literaturverzeichnis

Computergrafik / Visualisierung

Clark, J.H. (1976): Hierarchical Geometric Models for Visible Surface Algorithms. In: Communications of the ACM 19(10). 547-554. [8]

Coors, V.; Flick, S. (1998): Integrating Levels of Details in a Web-based GIS. In: Laurini, R. ; Makki, K. ; Pissinou, N. (Hrsg.): ACM-GIS '98, Proceedings of the 6th Internat. Symposium on Advances in Geographic Information Systems. WashingtonD.C. [8]

Döllner, J. ; Buchholz, H. ; Nienhaus, M. ; Kirsch, F. (2005): Illustrative Visualization of 3D City Models. In: Erbacher, R.F. ; Roberts, J.C. ; Grohn, M.T. ; Borner, K. (Hrsg.): Proceedings of Visualization and Data Analysis 2005 (Electronic Imaging 2005, January 16-20). San Jose, California. 42-51. [11]

Foley, J. ; van Dam, A. ; Feiner, S. ; Hughes, J. (1995): Computer Graphics: Principles and Practice. 2.Edition. Addison Wesley, Boston. [15]

Jochem, R. (2012): Neue 3D-Internet-Technologien für zukünftige Geoanwendugen im Netz. In: gis Trends & Markets 1. Wichmann, Heidelberg. 16-23.
(New 3D internet technologies for future georelated webapps) [20]

Juen, G. ; Steffens, A. (2012): Internet Visualisierung von 3D-Stadtmodellen. Vortrag am 21.03.2012. 3D-Forum. Lindau. Verfügbar unter: www.citygml.de/tl_files/citygml/public/LindauNeu.pdf [19]

Hoppe, H. (1996): Progressive meshes. In: ACM SIGGRAPH Proceedings. 99-108. [8]

Mao, B. (2011): Visualisation and Generalisation of 3D City Models. Dissertation. Stockholm. [20]

Datenbanken

Ahrends, J. ; Lenz, D. ; Schwanke, P. ; Unbescheid, G. (2011a): Oracle 11g Release 2 für den DBA. Produktive Umgebungen effizient konfigurieren, optimieren und verwalten. Addison-Wesley, München. [22] [42] [55]

Ahrends, J. (2011b): Oracle Standard Edition als Alternative zur Enterprise Edition. Vortrag am 16.11.2011. DOAG Konferenz. Nürnberg. Verfügbar unter: http://www.carajandb.com/vortraege.htm [2]

Aitchinson, A. (2009): Beginning Spatial with SQL Server 2008. Apress, New York. [24]

Aitchinson, A. (2012): Pro Spatial With SQL Server 2012. Apress, New York. [24]

Botha, I. ; Velicanu, A. ; Bâra, A. (2011): Modeling Spatial Data within Object Relational-Databases. In: Database Systems Journal vol.II, 1/2011. Economic Informatiocs Department, Bucharest. 33-40. [21]

Brendel, J-C. (2011): PostgreSQL oder MySQL: Wer skaliert besser? - Das Datenbank-duell. In: ADMIN – Netzwerk und Security. 6/2011. 40-45. [55]

Breunig, M. ; Zlatanova, S (2011): 3D geo-database research: retrospective and future directions. In: Computers & Geosciences: an international journal 37(7). 791-803. [25] [39]

Brewer, E.A. (2000): Towards robust distributed systems. Vortrag am 19.07.2000. Nineteenth ACM Symposium on Principles of Distributed Computing (PODC). Portland. Verfügbar unter: http://www.cs.berkeley.edu/~brewer/cs262b2004/PODC-keynote.pdf [25]

Brinkhoff, T. (2008): Geodatenbanksysteme in Theorie und Praxis. Einführung in objektrelationale Geodatenbanken unter besonderer Berücksichtigung von Oracle Spatial. 2.Auflage. Wichmann, Heidelberg. [1] [21] [22] [24] [37] [39] [46] [70]

Choquette, Y. (2010): Moving from Oracle/ArcGIS to PostgreSQL/PostGIS. Vortrag am 08.09.2010. FOSS4G. Barcelona. Verfügbar unter: http://2010.foss4g.org/presentations/3155.pdf [33]

Codd, E.F. (1970): A Relational Model of Data for Large Shared Data Banks. In: Communications of the ACM, 13(6). 377-387. [21]

Courtin, O. (2011): PostGIS meets the Third Dimension. Vortrag am 30.05.2011. Geomatics Open Lectures Series. Delft. Verfügbar unter: http://www.slideshare.net/SimeonNedkov/postgis-3d-implementation [23] [26]

Darold, G. (2011): Best practices with Ora2Pg. Vortrag am 19.10.2011. PGConf.EU. Amsterdam. Verfügbar unter: http://ora2pg.darold.net/ora2pg-best-practices.pdf [41]

Edlich, S. ; Friedland, A. ; Hampe, J. ; Brauer, B. ; Brückner, M. (2011): NoSQL. Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken. 2.Auflage. Hanser, München. [25]

Eisentraut, P. ; Helmle, B. (2011): PostgreSQL Administration. 2.Auflage. O'Reilly, Köln. [55]

Fiorillo, C. (2012): Oracle Database 11gR2 Performance Tuning Cookbook. Packt, Birmingham. [53]

Fraysse, L. (2009): Développement d'une geodatabase CityGML à partir de solutions open source. Praktikumsbericht – unveröffentlicht, Arles. [26] [33]

Fröhlich, L. (2009): Oracle 11g – Das umfassende Handbuch. MITP, Frechen. [54]

Godfrind, A. (2009): Oracle's Point Cloud datatype. Vortrag am 26.11.2009. Management of massive point cloud data: wet and dry. Delft. Verfügbar unter: http://www.ncg.knaw.nl/Studiedagen/09PointClouds/programma.html [22]

Guttman, A. (1984): R-Trees: A dynamic index structure for spatial searching. In: Proceedings of the ACM SIGMOD. Boston. 47–57. [39]

Haberl, U. ; Müllenbach, S. (2009): Bildern & Datenbanken. Eine pragmatisch orientierte Bestandsaufnahme. Augsburg. Verfügbar unter: http://www.hs-augsburg.de/~watzlaff. [34]

Hagander, M. (2007): Advanced PostgreSQL on Windows. Vortrag am 23.05.2007. PGCon. Ottawa. Verfügbar unter: http://www.hagander.net/talks/Advance%20PostgreSQL%20on%20Windows.pdf [58]

Hellerstein, J.M. ; Naughton, J.F. ; Pfeffer, A. (1995): Generalized search trees for database systems. In: Proceedings 21st VLDB(Very Large Data Bases) Conference. Zürich. 562–573. [39]

Kalberer, P. (2010): SpatiaLite, das Shapefile der Zukunft?. Vortrag am 03.03.2010. FOSSGIS. Osnabrück. Verfügbar unter: http://www.fossgis.de/konferenz/2010/events/127.de.html [24]

Kolbe, T.H. ; König, G. ; Nagel, C. ; Stadler, A. (2009): 3D-Geo-Database for CityGML. Version 2.0.1. Documentation. Berlin. Verfügbar unter: http://www.3dcitydb.net/index.php?id=1897 [2] [15] [16] [17] [21] [29]

Kolbe, T.H. ; Nagel, C. ; Herreruela, J. (2012): 3D-Geo-Database for CityGML. Addendum to the 3D City Database Documentation Version 2.0. Berlin. Verfügbar unter: http://www.3dcitydb.net/index.php?id=1897 [20]

Kothuri, R. ; Godfrind, A. ; Beinat, E. (2007): Pro Oracle Spatial for Oracle Database11g. Appres, New York. [22] [37]

KUNDE, F ; NAGEL, C ; HERRERUELA, J ; KÖNIG, G. ; KOLBE, T.H. ; ASCHE, H. (2012): 3D City Database for CityGML. 3D City Database Version 2.0.6-postgis - Importer/Exporter Version 1.4.0-postgis. Tutorial. Verfügbar unter: http://opportunity.bv.tu-berlin.de/software/documents/63 [297]

NAGEL, C. ; STADLER, A. (2008): Die Oracle-Schnittstelle des Berliner 3D-Stadtmodells. In: CLEMEN, C. (Hrsg.): Entwicklerforum Geoinformationstechnik 2008. Shaker, Aachen. 197-221. [16] [17] [18] [44] [45] [*295*]

OBE, R.O. ; HSU, L. (2010): PostGIS in Action. Manning, New York. [23] [24]

OPSAHL, L. ; BJØRKELO, K. ; NYSTUEN, I. (2012): Migrating from Oracle/ArcSDE to PostgreSQL/PostGIS: Synchronization of geodatabase content during transition period. Vortrag am 23.05.2012. EuroSDR Workshop on PostGIS DBMS. Southhampton. Verfügbar unter: http://www.eurosdr.net/workshops/PostGIS/2_Nystuen_Opsahl_Bjorkelo_Norwegian_Forest_and_Landscape_Institute.pdf [33]

PFEIFFER, T. ; WENK, A. (2010): PostgreSQL: Installation, Grundlagen, Praxis. Galileo, Bonn.

PLÜMER, L. ; GRÖGER, G. ; KOLBE, T.H. ; SCHMITTWILKEN, J. ; STROH, V. ; POTH, A. ; TADDEO, U. (2005): 3D Geodatenbank Berlin, Dokumentation V1.0 Institut für Kartographie und Geoinformation der Universität Bonn (IKG), lat/lon GmbH. Verfügbar unter: http://www.businesslocationcenter.de/imperia/md/content/3d/dokumentation_3d_geo_db_berlin.pdf [29]

RACINE, P. (2011): Store, manipulate and analyze raster data within the PostgreSQL/PostGIS spatial database. Vortrag am 16.09.2011. FOSS4G. Denver. Verfügbar unter: http://trac.osgeo.org/postgis/wiki/WKTRaster [23]

RIGGS, S. ; KROSING, H. (2010): PostgreSQL 9 Administration Cookbook. Packt, Birmingham. [23]

ROSENSTEEL, K. (2011): Large Customers Want PostgreSQL Too!. Vortrag am 24.03.2011. PostgreSQL Conference East (PGeast). New York. Verfügbar unter: http://www.postgresmigrations.com/files/4286158/uploaded/Large%20Customers%20Want%20PostgreSQL%20Too!.pdf *[41]*

SCHERBAUM, A. (2009): PostgreSQL – Datenbankpraxis für Anwender, Administratoren und Entwickler. Open Source Press, München. [23]

SIEBEN, J. (2010): Oracle PL/SQL. Das umfassende Handbuch. Galileo, Bonn. [41]

SINGH, G. (2011): Oracle to Postgres Migration – Considerations, Hurdles, and possible Solutions. Vortrag am 19.05.2011. PGCon. Ottawa. Verfügbar unter: ww.pgcon.org/2011/schedule/attachments/205_Oracle_to_Postgres_Migration.pdf [34]

SKULSCHUS, M. ; WIEDERSTEIN, M. (2011): Oracle PL/SQL. Comelio Medien, Berlin. [39]

SMITH, G. (2010): PostgreSQL 9.0 High Performance. Packt, Birmingham. [55]

SONDERMANN, B. (2008): Eine XML-Datenbank für Aristoteles3D. In: CLEMEN, C. (Hrsg.): Entwicklerforum Geoinformationstechnik 2008. Shaker, Aachen. 107-117. [25]

Stadler, A. ; Nagel, C. ; König, G. ; Kolbe, T.H. (2009): Making interoperability persistent: A 3D geo database based on CityGML. In: Lee, J. ; Zlatanova, S. (Hrsg.): 3D Geoinformation Sciences. Lecture Notes in Geoinformation and Cartography. Springer, Berlin / Heidelberg. 175-192.

Vohra, D. (2008): JDBC 4.0 and Oracle JDeveloper for J2EE Development. Packt, Birmingham. [30]

Yeung, A.K.W. ; Hall, G.B. (2007). Spatial database systems. Design, Implementation and Project Management. Springer, Heidelberg. [39]

Geo- / Informatik

Bartelme, N. (2005): Geoinformatik. Modelle Strukturen Funktionen. 4.Auflage. Springer, Heidelberg. [1] [21]

Bill, R. (2010): Grundlagen der Geo-Informationssysteme. 5.Auflage. Wichmann, Heidelberg. [1]

Clementini, E., Di Felice, P. (1994): A Comparison of Methods for Representing Topological Relationships. In: Information Sciences 80. 1-34. [249]

Edmead, M.T.; Hinsberg, P. (1998): Windows NT Performance Monitoring, Benchmarking, and Tuning. New Riders, Indianapolis. [58]

Egenhofer M.J. ; Herring J (1990): A mathematical framework for the definition of topological relationships. Fourth International Symposium on Spatial Data Handling. Zürich. 803-813. [249]

Ellul, C. ; Haklay, M.M. (2009): Using a B-Rep Structure to Query 9-Intersection Topological Relationships in 3D GIS – Reviewing the Approach and Improving Performance. In: Lee, J. ; Zlatanova, S. (Hrsg.): 3D Geoinformation Sciences. Lecture Notes in Geoinformation and Cartography. Springer, Berlin / Heidelberg. 127-151. [15]

Friedman, M ; Pentakalos, O.(2002): Windows 2000 Performance Guide. O'Reilly, Sebastopol. [58]

Gerschwitz, A. ; Gruber, U. ; Schlüter, S. (2011): Dir dritte Dimension im ALKIS. In: Kummer, Frankenberger (Hrdg.): Das deutsche Vermessungs- und Geoinformationswesen. Wichmann, Heidelberg. 279-310. [8] [12]

Goodchild, M. (2007). Citizens as sensors: the world of volunteered geography. GeoJournal, 69(4). 211-221. [11]

Köninger, A. ; Bartel, S. (1998): 3D-GIS for Urban Purposes. In:Geoinformatica 2(1). 79-103. [7] [8]

Lemmens, M. (2011): Geo-information. Technologies, Applications and the Environment. Springer, Dodrecht / Heidelberg. [9]

Neis, P. ; Zielstra, D. ; Zipf, A. ; Struck, A. (2010): Empirische Untersuchungen zur Datenqualität von OpenStreetMap - Erfahrungen aus zwei Jahren Betrieb mehrerer OSM-Online-Dienste. In: Strobl, J. ; Blaschke, T. ; Griesebner, G. (Hrsg.): Angewandte Geoinformatik 2010. Beiträge zum 22. AGIT-Symposium Salzburg. Wichmann, Heidelberg. 421-425. [11]

Zimmermann, A. (2012): Basismodelle der Geoinformatik. Hanser, München. [18] [21]

Java

Abst, D. (2010): Masterkurs Client/Server Programmierung mit Java. 3.Auflage. Vieweg & Teubner, Wiesbaden. [45]

Scholz, M. ; Niedermeyer, S. (2009): Java und XML – Grundlagen, Einsatz, Referenz. 2.Auflage. Galileo, Bonn. [44]

Ullenboom, C. (2012): Java ist auch eine Insel. 12.Auflage. Galileo, Bonn. [44]

Stadtmodelle - Erhebung

Baillard, C. ; Dissard, O. ; Maitre, H. (1998): Segmentation of Urban Scenes from Aerial Stereo Imagery. In: Proceedings CVPR. 1405-1407. [7]

Brenner, C. ; Haala, N. (1998): Rapid acquisition of virtual reality city models from multiple data sources. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 32(5). 323–330. [7]

Dorninger, P. ; Pfeifer, N. (2008): A Comprehensive Automated 3D Approach for Building Extraction, Reconstruction, and Regularization from Airborne Laser Scanning Point Clouds. In: Sensors 8/2008. 7323-7343. [10]

Fan, H. ; Meng, L. ; Jahnke, M. (2009): Generalization of 3D Buildings Modelled by CityGML. In: Lee, J. ; Zlatanova, S. (Hrsg.): 3D Geoinformation Sciences. Lecture Notes in Geoinformation and Cartography. Springer, Berlin / Heidelberg. 387-405. [19]

Goetz, M. ; Zipf, A. (2012a): The Evolution of Geo-Crowdsourcing: Bringing Volunteered Geo graphic Information to the Third Dimension. In: Sui, D ; Elwood, S. ; Goodchild, M.F. (Hrsg.): Volunteered Geographic Information, Public Participation, and Crowdsourced Production of Geographic Knowledge. Springer, Berlin. [11]

Goetz, M. ; Zipf, A. (2012b): Towards Defining a Framework for the Automatic Derivation of 3D CityGML Models from Volunteered Geographic Information . International Journal of 3-D Information Modeling (IJ3DIM), 1(2). IGI-Global. [18]

Grün, A., Kübler, O., Agouris, P. (Hrsg.) (1995): Automatic Extraction of Man-Made Objects from Aerial and Space Images. Birkhäuser, Basel. [7]

Gülich, E. (2005): Erfassung von 3D-Daten mit Digitaler Photogrammetrie. In: Coors,V. ; Zipf,A. : 3D-Geoinformationssysteme. Grundlagen und Anwendungen. Wichmann, Heidelberg. 4-22. [10]

Haala, N. (2009): Comeback of Digital Image Matching. In: Fritch, D. (Hrsg.): Photogrammetric Week '09. Wichmann, Heidelberg. 289-301. [10]

Haala, N. ; Kada, M. (2010): An update on automatic 3D building reconstruction. In: SPRS Journal of Photogrammetry and Remote Sensing 65. 570–580. *[9]*

Krückhans, M. ; Schmittwilken, J. (2009): Attributierte Grammatiken zur artifiziellen Verfeinerung von 3D-Stadtmodellen. In: DGPF-Jahrestagung, Tagungsband 18/2009. Jena. [11]

Ledoux, H. ; Meijers, M. (2009): Topologically consistent 3D city models obtained by extrusion. In: International Journal of Geographical Information Science 25(4). 557-574. [11]

Lieckfeldt, P. (2012): High-Quality 3D Modelling Using Georeferenced Photography. Verfügbar unter: http://www.gta-geo.de/images/pressemitteilungen/2012-pi13_gta_tridicon_3d_landmark_e.pdf [11]

Li, J. ; Guan, H. (2011): 3D building reconstruction from airborne LIDAR point clouds fused with arial imagery. In: Yang, X. (Hrsg.): Urban Remote Sensing: Monitoring, Synthesis and Modeling in the Urban Environment. Wiley-Blackwell, West-Sussex. 75-91. [9] [10]

Müller, P. ; Wonka, P. ; Haegler, S. ; Ulmer, A. ; Andvangool, L. (2006): Procedural Modeling of Buildings. In: Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics 25. ACM Press, New York. 614–623. [11]

Müller, P. ; Zeng, G. ; Wonka, P. ; Van Gool, L. (2007): Image-based Procedural Modeling of Facades. In: ACM Transactions on Graphics 26(3). Article 85. [11]

Pu, S. ; Vosselman, G. (2009): Knowledge based reconstruction of building models from terrestrial laser scanning data. In: ISPRS Journal of Photogrammetry and Remote Sensing 64(6). 575-584. [11]

Richter, R. ; Döllner, J. (2011): Integrated real-time visualization of massive 3D point clouds and geo-referenced texture data - Integrierte Echtzeit-Visualisierung von massiven 3D- Punktwolken und georeferenzierten Texturdaten. In: Photogrammetrie - Fernerkundung – Geoinformation 3. 145-154. [10]

Schilcher, M. ; Guo, Z. ; Klaus, M. ; Roschlaub, R. (1999): Aufbau von 3D-Stadtmodellen auf der Basis von 2D-GIS. In: Zeitschrift für Photogrammetrie und Fernerkundung 67(3). 157-170. [8]

Yin, X. ; Wonka, P. ; Razdan, A. (2009): Generating 3D Building Models from Architectural Drawings: A Survey. In: Computer Graphics and Applications, IEEE 29(1). 20-30. [11]

Stadtmodelle - Anwendung

Agugiaro G., Remondino F., Girardi G., von Schwerin J., Richards-Rissetto H., De Amicis R. (2011): A Web-based Interactive Tool for Multi-Resolution 3D models of a Maya Archaeological Site. In: 4th ISPRS International Workshop 3D-ARCH 2011: "3D Virtual Reconstruction and Visualization of Complex Architectures" 2-4 March 2011. Tagungsband. Trento.
Verfügbar unter: http://mayaarch3d.unm.edu/publications.php [26]

Bauer, W. ; Mohl, H-U. (2005): Das 3D-Stadtmodell der Landeshauptstadt Stuttgart. In: Coors, V. ; Zipf, A.: 3D-Geoinformationssysteme. Grundlagen und Anwendungen. Wichmann, Heidelberg. 265-278. [7]

Casper, E. (2008): Geodatenmanagement in der Berliner Verwaltung – Amtliches 3D Stadtmodell für Berlin. Projektbericht 2007. Verfügbar unter: http://www.businesslocationcenter.de/imperia/md/content/3d/efre_ii_projektdokumentation.pdf [29]

Czerwinski, A. ; Gröger, G. ; Dörschlag, D. ; Stroh, V. ; Kolbe, T.H. ; Plümer, L. (2008): Nachhaltige Erweiterung der Geodateninfrastruktur für 3D-Geodaten auf Basis von CityGML – am Beispiel der EU-Umgebungslärmkartierung. In: Kartographische Schriften 14. 67-74. [12] [14]

de Laat, R. ; van Berlo, L. (2011): Integration of BIM and GIS: The Development of the CityGML GeoBIM Extension. In: Kolbe, T.H. ; König, G. ; Nagel, C. (Hrsg.): Advances in 3D Geo-Information Sciences. Lecture Notes in Geoinformation and Cartography. Springer, Berlin / Heidelberg. 211-225. [14]

Döllner, J. ; Kolbe, T.H. ; Liecke, F. ; Sgouros, T. ; Teichmann, K. (2006): The Virtual 3D City Model of Berlin - Managing, Integrating and Communicating Complex Urban Information. In: Proceedings of the 25th Urban Data Management Symposium UDMS 2006 in Aalborg, Denmark, May 15-17. [11]

El-Mekawy, M. ; Östman, A. ; Shahzad, K. (2011): Towards Interoperating CityGML and IFC Building Models: A Unified Model Based Approach. In: Kolbe, T.H. ; König, G. ; Nagel, C. (Hrsg.): Advances in 3D Geo-Information Sciences. Lecture Notes in Geoinformation and Cartography. Springer, Berlin / Heidelberg. 73-93. [18]

Gröger, G. ; Benner, J. ; Dörschlag, D. ; Drees, R. ; Gruber, U. ; Leinemann, K.; Löwner, M-O. (2005): Das interoperable 3D-Stadtmodell der SIG 3D. In: zfv – Zeitschrift für Geodäsie, Geoinformation und Landmanagement 6. Wißner Verlag, Augsburg. [7] [8]

Hijazi, I. ; Ehlers, M. ; Zlatanova, S. ; Becker, T. ; van Berlo, L. (2011): Initial Investigations for Modeling Interior Utilities Within 3D Geo Context: Transforming IFC-Interior Utility to CityGML/UtilityNetworkADE. In: Kolbe, T.H. ; König, G. ; Nagel, C. (Hrsg.): Advances in 3D Geo-Information Sciences. Lecture Notes in Geoinformation and Cartography. Springer, Berlin / Heidelberg. 95-113. [14]

Isikdag, U. ; Zlatanova, S. ; Underwood, J. (2012): An opportunity analysis on the future role of BIMs in urban data management. In: Zlatanova, S. ; Ledoux, H. ; Fendel, E. ; Rumor, M. (Hrsg.): Urban and regional data management. UDMS Annual 2011. Taylor & Francis, CRC Press, Balkema, London / Leiden. 25-36. [11]

KRÄMER, M ; HOPF, C. (2012): Gewinnbringende Nutzung von 3D-Stadtmodellen: Ein Erfahrungsbericht aus Mainz. In: gis Trends & Markets 1/2012. Wichmann, Heidelberg. 28-35. (Profitable use of 3D city models: The Mainz Experience) [7]

KOLBE, T.H. (2004): Interoperable 3D-Visualisierung („3D Web Map Server"). In: Symposium Praktische Kartographie 2004 in Königslutter, 17-19.05.2004. Tagungsband. Kartographische Schriften 9. Kirschbaum Verlag, Bonn. [7] [8]

KOLBE, T.H. (2009): Representing and Exchanging 3D City Models with CityGML. In: Lee, J. ; Zlatanova, S. (Hrsg.): 3D Geoinformation Sciences. Lecture Notes in Geoinformation and Cartography. Springer, Berlin / Heidelberg. 15-31. [1] [14]

KOLBE, T.H. (2011): Urban Modelling. Übersicht und Vergleich der Konzepte für die dreidimensionale Modellierung urbaner Information. Vortrag am 12.01.2011. Ringvorlesung Geoinformatik. Potsdam. [20]

KULAWIK, R. ; SCHILLING, A. ; ZIPF, A. (2009): Landesweite 3D-Stadtmodelle im Internet auf Basis offener Standards des Open Geospatial Consortiums (OGC) - das Beispiel Nordrhein-Westfalen 3D. In: SCHRENK, M. ; POPOVICH, V.V. ; ENGELKE, D. ; ELISEI, P. (Hrsg.): REAL CORP 2009: CITIES 3.0 – Smart, Sustainable, Integrative. 14th International Conference on Urban Planning, Regional Development and Information Society. Tagungsband. 293-302. [18] [19]

LUDWIG, D. ; MCKINLEY, L. (2010): Solar Atlas of Berlin. In: GIM International 24(3). Verfügbar unter: http://www.gim-international.com/issues/articles/id1504-Solar_Atlas_of_Berlin.html. [12]

RANDT, B. ; BILDSTEIN, F. ; KOLBE, T.H. (2007): Use of Virtual 3D Landscapes for Emergency Driver Training. In: Proc. of the Int. Conference on Visual Simu-ation IMAGE. Scottsdale. [12]

STADLER, A. (2008): Kohärenz von Geometrie und Semantik in der Modellierung von 3D Stadtmodellen. In: CLEMEN, C. (Hrsg.): Entwicklerforum Geoinformationstechnik 2008. Shaker, Aachen. 167-181. [13] [15]

STOTER, J. ; VAN DEN BRINK, L. ; VOSSELMAN, G. ; GOOS, J. ; ZLATANOVA, S. ; VERBREE, E. ; KLOOSTER, R. ; VAN BERLO, L. ; VESTJENS, G. ; REUVERS, M. ; THORN, S. (2011): A generic approach for 3D SDI in the Netherlands, In: Proceedings of the Joint ISPRS Workshop on 3D City Modelling & Applications and the 6th 3D GeoInfo Conference, 26-28 June. Wuhan. Verfügbar unter: http://www.gdmc.nl/zlatanova/publications.htm [12] [20]

Standards

ANDRAE, C. (2008): OpenGIS essentials: Spatial Schema – ISO 19107 und ISO 19137 vorgestellt und erklärt. Wichmann, Heidelberg. [15]

CADUFF, D. ; NEBIKER, S. (2010): 3D-Standards und Web-Services. In: Geomatik Schweiz 9. 397-401. [13] [20]

GEOINFODOK (2009): Dokumentation zur Modellerung der Geoinformationen des amtlichen Vermessungswesens. Hauptdokument Version 6.0.1. Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland. [2] [12]

Gröger, G. ; Kolbe, T.H. ; Czerwinski, A. ; Nagel, C. (2008): OpenGIS® City Geography Markup Language (CityGML) Encoding Standard. Version 1.0.0. OGC 08-007rl. [13]

Gröger, G. ; Kolbe, T.H. ; Nagel, C. ; Häfele, K-H. (2012): OGC City Geography Markup Language (CityGML) Encoding Standard. Version 2.0.0. OGC 12-019. [18]

Hagedorn, B. (2010): Web View Service Discussion Paper. Version 0.3.0. OGC 09-166r2. [20]

Herring, J. (2001): The OpenGIS Abstract Specification, Topic 1: Feature Geometry (ISO 19107 Spatial Schema). Version 5. OGC 01-101. [8]

Herring, J. (2010): OpenGIS Implementation Standard for Geographic information – Simple feature access – Part 2: SQL option. Version 1.2.1. OGC 06-104r4. [21]

Herring, J. (2011): OpenGIS Implementation Standard for Geographic information – Simple feature access – Part 1: Common architecture. Version 1.2.1. OGC 06-103r4. [13]

INSPIRE Thematic Working Group Building (TWG BU) (2011): INSPIRE Data Specification on Building – Draft Guidelines. Version D2.8.III.2. [12]

ISO – International Organisation for Standardization (2003a): ISO 19107:2003. Geographic Information – Spatial Schema. [15]

ISO – International Organisation for Standardization (2003b): Database Languages – SQL, ISO/IEC 9075-*:2003. [23] [40]

Lake, R. ; Cuthbert, A. ; O'Rourke, B. ; Keighhan, E. ; Margoulies, S. ; Sharma, J. ; Daisey, P. ; Johnson, S. (2000): Geography Markup Language (GML). Version 1.0. OGC 00-029. [13]

Portele, C. (2010): Geography Markup Language (GML) – Extended schemas and encoding rules. Version 3.3.0. OGC 10-129r1. [13]

Schilling, A. ; Kolbe, T.H. (2010): Draft for Candidate OpenGIS Web 3D Service Interface Standard. Version 0.4.0. OGC 09-104r1. [20]

Internetquellen

www1	http://www.citygml.org/ [1]
www2	http://www.opengeospatial.org/standards/citygml [1]
www3	http://www.3dcitydb.net/ [2]
www4	http://www.sig3d.de/ [8]
www5	http://www.virtualcitysystems.de/5-produkte/buildingreconstruction.html [10]
www6	http://www.tridicon.de/ [10]
www7	http://www.moss.de/deutsch/produkte/novafactory/modul-3d-pro/ [10]
www8	http://www.realitymaps.de/staedte/leistungen/datenerfassung.html [10]
www9	http://www.esri.com/software/cityengine/index.html [11]
www10	http://www.osm-3d.org/map.htm [11]
www11	http://wiki.openstreetmap.org/wiki/3D_Development [11]
www12	http://www.adv-online.de [12]
www13	http://www.opengeospatial.org/standards/gml [13]
www14	http://de.wikipedia.org/wiki/Virtual_Reality_Modeling_Language [14]
www15	http://www.agiledata.org/essays/impedanceMismatch.html [15]
www16	http://www.citygmlwiki.org/index.php/Main_Page [18]
www17	http://bimserver.org/ [18]
www18	http://www.geores.de/geoResProduktePlugins.html [19]
www19	http://www.citygml.de/index.php/Download.html [19]
www20	http://atlandsend.typepad.com/at-lands-end/2011/08/the-future-of-landxplorer-studio-professional.html [19]
www21	http://www.webglearth.org/ [20]
www22	http://www.oracle.com/technetwork/indexes/downloads/index.html [22]
www23	http://www.postgresql.org/ [22]
www24	http://blog.jelastic.com/2012/07/23/software-stack-market-share-july-2012-2/ [22]
www25	http://www.postgresql.org/community/ [23]
www26	http://de.wikipedia.org/wiki/BSD-Lizenz [23]
www27	http://www.enterprisedb.com/ [23] [41]
www28	http://postgis.org [23]
www29	http://trac.osgeo.org/geos/ [23]
www30	http://www.gdal.org/ [23]
www31	http://trac.osgeo.org/proj/ [23]
www32	http://postgis.org/support/ [24]
www33	http://www.gnu.de/documents/gpl.de.html [24]
www34	http://www-01.ibm.com/software/data/spatial/ [24]
www35	http://workshops.opengeo.org/postgis-spatialdbtips/introduction.html [24]
www36	http://dev.mysql.com/doc/refman/5.0/en/spatial-extensions.html [24]

www37	http://mariadb.org/ [24]
www38	http://www.gaia-gis.it/gaia-sins/ [24]
www39	http://resources.arcgis.com/content/geodatabases/10.0/arcsde-installation-guides [24]
www40	https://github.com/couchbase/geocouch/ [25]
www41	https://github.com/neo4j/spatial [25]
www42	http://www.cpa-systems.de/index.php?do=pro&do2=3d&lang=d [26]
www43	http://www.snowflakesoftware.com/products/goloader/ [26]
www44	http://www.cityserver3d.de/ [26]
www45	http://www.businesslocationcenter.de/de/3d-stadtmodell [29]
www46	http://wiki.postgresql.org/wiki/Oracle_to_Postgres_Conversion [33]
www47	http://www.enterprisedb.com/resources-community/tutorials-quickstarts/all-platforms/migrating-oracle-postgres-plus-advanced-serv [33]
www48	http://www.postgresmigrations.com/more-detail-postgresql [33]
www49	http://docs.oracle.com/cd/B28359_01/appdev.111/b28400/sdo_objrelschema .htm#i1004087 [35]
www50	http://docs.oracle.com/cd/B28359_01/appdev.111/b28398/geor_intro.htm [37]
www51	http://www.postgis.org/documentation/manual-svn/using_raster.xml.html [37]
www52	http://www.postgresql.org/docs/9.1/interactive/plpgsql-porting.html [42]
www53	http://postgis.refractions.net/documentation/javadoc/org/postgis/package-summary.html [44]
www54	http://jdbc.postgresql.org/development/privateapi/index.html [44]
www55	http://opportunity.bv.tu-berlin.de/software/projects/3dcitydb-imp-exp/ [46]
www56	http://docs.oracle.com/cd/B19306_01/appdev.102/b14373/oracle/spatial/geometry/JGeometry.html [47]
www57	https://wiki.postgresql.org/wiki/Performance_Analysis_Tools [58]
www58	http://kappasys.net/cms/index.php?id=23&L=5%60 [74]
www59	http://www.iscopeproject.net/ [75]

Anhang:

www60	http://opportunity.bv.tu-berlin.de/software/documents/63 [99]
www61	http://commons.apache.org/dbcp/ [243]
www62	http://tomcat.apache.org/index.html [243]
www63	http://www.mchange.com/projects/c3p0/ [243]
www64	http://spatialreference.org/ [247]

A b b i l d u n g s v e r z e i c h n i s

Abbildung 1: Die LOD-Stufen am Beispiel von Gebäuden des Berliner Stadtmodells *[9]*

Abbildung 2: Automatisierte Ableitung von Gebäuden aus einem DEM (Auschnitt von BuidingReconstruction, einer Software von virtualcitySYSTEMS) *[10]*

Abbildung 3: Komplexes Objekt mit voll kohärenter räumlich-semantischer Struktur [Quelle: Stadler 2008] *[13]*

Abbildung 4: Aufbau des CityGML-Anwendungsschemas *[14]*

Abbildung 5: Vereinfachung des Geometriemodells (*3DCityDB*-Doku. 2009: 18) *[16]*

Abbildung 6: Gegenüberstellung von UML-Model und Datenbankschema am Beispiel des Transportation-Modells (*3DCityDB* Doku. 2009: 32, 64) *[17]*

Abbildung 7: Tag-Cloud mit Namen von Softwareprodukten, die CityGML verwenden *[19]*

Abbildung 8: Marktanteil von Open-Source-Datenbanken *[22]*

Abbildung 9: Aufbau der 3DCityDB *[30]*

Abbildung 10: Ordnerübergreifende Relationen zwischen Tabellen *[31]*

Abbildung 11: Durchführungsplan der Portierung *[32]*

Abbildung 12: Indizierung von Geometrien mittels ihrer Bounding Boxes [Quelle: wikipedia] *[38]*

Abbildung 13: Aufbau einer PostgreSQL Datenbank *[42]*

Abbildung 14: Prozessketten beim Im- und Export von CityGML-Daten [vgl. Nagel, Stadler 2008] *[45]*

Abbildung 15: GUI des Importer/Exporter *[46]*

Abbildung 16: Importweg der Geometrien in Java *[48]*

Abbildung 17: Exportweg der Geometrien in Java *[49]*

Abbildung 18: Im- und Export von Texturen *[49]*

Abbildung 19: Ausgewählte Datensätze im Überblick *[56] [57]*

Abbildung 20: Testergebnisse des Datenimports mit verschieden Datensätzen, variierender Anzahl paralleler Ausführungsthreads und jeweils aktivierten und deaktivierten Indizes *[61]*

Abbildung 21: Testergebnisse des Datenexports bei variierender Anzahl paralleler Ausführungsthreads *[65]*

Abbildung 22: Speicherbedarf der Datensätze in Oracle und PostgreSQL *[67]*

Abbildung 23: Relative zeitliche Verlängerung bei Importen bei Hinzunahme von Texturen und bei gleichzeitiger Indizierung der Daten *[70]*

Anhang:

Abbildung D1: Splash-Screen der PostGIS-Version des Importer/Exporter *[285]*

Abbildung D2: Datenbank-Panel des Importer/Exporter mit Log-Beispielen für das erfolgreiche Verbinden mit der Datenbank und Rückmeldungen beim Deaktivieren der Indizes *[286]*

Abbildung D3: Import-Panel während eines Importes mit Log-Eintrag im Konsolenfenster für einen erfolgreich abgeschlossenen Import *[287]*

Abbildung D4: Export-Panel mit Log-Eintrag im Konsolenfenster für einen erfolgreich abgeschlossenen Export *[288]*

Abbildung D5: Das KML-Exporter-Panel *[289]*

Abbildung D6: Das MapWindow zum Betrachten und Editieren des gegenwärtigen Bounding Box Filters *[289]*

Abbildung D7: Einstellungsmöglichkeiten für die verschiedenen Module des Importer/Exporter *[290]*

Abbildung D8: Datensatz Stadt Ettenheim (verfügbar unter citygml.org) im FZKViewer *[291]*

Abbildung D9: Datensatz Stadt Frankfurt (verfügbar unter citygml.org) im LandXplorer *[292]*

Abbildung D10: Datensatz München (verfügbar unter citygml.org) in gvSIG mit 3D-Erweiterung und CityGML-Patch *[292]*

Abbildung D11: Das geographische Institut der Universität Potsdam mit Balloon-Template in Google Earth (aus dem 3D-Stadtmodell Potsdam, Eigentum der Stadt Potsdam in Kooperation mit der virtualcitySYSTEMS GmbH) *[293]*

Abbildung D12: Die vier Arten des KML-Export (Footprint, Extruded, Geometry, COLLADA) am Beispiel des Gasometer-Datensatzes (Teil des Release) *[293]*

Abbildung D13: Visuelle Analyse von Ergebnissen des Matching/Merging-Plugins mit dem FME Inspector (aus dem 3D-Stadtmodell Potsdam, Eigentum der Stadt Potsdam in Kooperation mit der virtualcitySYSTEMS GmbH) *[294]*

Abbildung D14: Prozessketten beim Im- und Export von CityGML-Daten [Quelle: Nagel, Stadler 2008] *[295]*

Abbildung D15: Angefertigte Skizze, um einen ersten „Überblick“ von der Softwarearchitektur des Importer/Exporter zu erhalten *[296]*

Tabellenverzeichnis

Tabelle 1: 3D Austauschformate für Stadtmodelle im Vergleich (vgl. Kolbe 2011: 59; Caduff, Nebiker 2010: 400; Stoter et al. 2011: 14) [20]

Tabelle 2: Unterschiede bei den Datentypen [34]

Tabelle 3: Unterschiede bei den Geometrie-Datentypen [35]

Tabelle 4: Unterschiedliche Namen, gleiche Funktion [50]

Tabelle 5: Änderungen in postgresql.conf [55]

Tabelle 6: Überwachte Systemparameter [vgl. Edmead, Hinsberg 1998; Friedman, Pentakalos 2002] [58]

Tabelle 7: Auswirkung auf die Importzeit bei aktivierten gegenüber deaktivierten Indizes [63]

Tabelle 8: Verhältnis Importzeit gegenüber ansteigender Datenmenge und -fläche [63]

Tabelle 9: Auswirkungen auf die Importzeit bei Hinzunahme von Texturen [64]

Tabelle 10: Auswirkungen auf die Zeit beim zusätzlichen Schreiben von Texturdateien [66]

Tabelle 11: Verhältnis Exportzeit gegenüber ansteigender Datenmenge und -fläche [66]

Anhang:

Tabelle A1: Ordnerstruktur des 3DCityDB-postgis Installations-Paketes [99]

Tabelle A2: Unterschiede bei den Top-Level-Dateien des 3DCityDB Installations-Paketes [100]

Tabelle A3: Der Inhalt vom Schema-Ordner im Überblick [109] [110]

Tabelle A4: Der Inhalt vom UTIL-Ordner im Überblick [139]

Tabelle A5: Übersicht von PL/pgSQL Skripten [142]

Tabelle C1: Zeiten beim Import von Datensätzen [282]

Tabelle C2: Zeiten beim Export von Datensätzen [283]

Tabelle C3: Speicherplatzbedarf in der Datenbank in MB [*284*]

Fremdwörterverzeichnis

In diesem Verzeichnis sind alle Fremdwörter aufgeführt, die im Text nicht näher erläutert wurden. Die Seitenzahlen verweisen nur auf das erste Auftreten des Wortes im Text.

Fremdwort	Erklärung
Array	Ein Feld in dem mehrere Variablen eines Datentyps abgespeichert und indiziert werden können, z.B. Feld = (1, 2, 3 etc...) *[41]*
B-Baum / R-Baum	Art der Baumstruktur eines Datenbank-Indexes *[38]*
Batch-Datei	führt ein für Microsoft DOS geschriebenes Stapelverarbeitungsskript aus, und minimiert die Arbeit auf Kommandozeile *[39]*
Best Practice	Bezeichnung für bewährte Methoden *[29]*
Benchmarking	Testverfahren, dass die Leistungsgrenzen von u.a. Software ermittelt *[55]*
Bounding Box	minimal umschließendes Rechteck eines räumlichen Objektes *[38]*
Change Request	Änderungsvorschlag an einen bestehenden Standard *[18]*
Default	Im Vorhinein festgelegter Wert, mit dem die weiteren Prozessschritte garantiert funktionieren *[39]*
Exception	Ausnahmesituation beim Abarbeiten von Prozessen *[50]*
Facilitymanagement	Verwaltung und Bewirtschaftung von Gebäuden, Anlagen etc. *[8]*
Framework	umschreibt eine Gruppe von bestimmten Werkzeugen, mit denen Programmierer eine Anwendung erstellen können *[18]*
Grid Computing	lose Koppelung von mehreren Computern, die gemeinsam einen virtuellen Supercomuputer erzeugen, der rechenintensive Prozesse verarbeitet *[22]*
InputStream	Java-Klasse, die einen kontinuierlichen Datenstrom ermöglicht, bei dem der Umfang nicht im Voraus bekannt ist *[49]*
Join	verknüpft zwei Datenbanktabellen miteinander *[16]*
Log	Protokoll, dass eine Software über der verarbeiteten Prozesse schreiben kann *[47]*
Mailing-List	öffentlicher Austausch von E-Mails einer bestimmten Gruppe *[23]*
mappen	bezeichnet u.a. das Einfügen von neuen Geodaten in OpenStreetMap *[11]*

Master/Slave	Mehrere Server teilen sich ein gemeinsame Ressource. Jener der als Master definiert ist kann uneingeschränkt auf die Ressource zugreifen. Die Slaves müssen erst vom Master angesprochen werden, um mit der Ressource zu interagieren. *[24]*
Middleware	ein neutrales Programm, dass zwischen zwei Anwendungen vermittelt *[24]*
Monitoring	das Überwachen von Prozessen *[52]*
Multithreading	Nebenläufigkeit von Verarbeitungssträngen *[44]*
MyISAM	Name der Software-Komponente („storage engine") des DBMS MySQL. Seit MySQL 5.5 wird die Engine InnoDB verwendet, das jedoch keine räumlichen Datentypen unterstützt *[24]*
Plugin	eigenständiges Erweiterungsmodul für eine existierende Anwendung *[46]*
Rendering	Erstellung einer Grafik aus einem Modell *[8]*
Repository	ein Verzeichnis zur Speicherung und häufig auch temoralen Verwaltung von Objekten, z.B. von einer Software *[46]*
Simple Features	eine Spezifikation des OGC über die Speicherung und den Zugriff auf Geodaten. Das geometrischen Klassenmodell umfasst Punkt, Linie, Polygon, ihre entsprechende Mehrfachauspägung („Multi") und eine Sammlung verschiedener Geomertien („GeometryCollection") *[13]*
Troubleshooting	Optimierung eines Produktes orientiert sich an ermittelten Problemen *[54]*
type modifier	das Modifizieren bzw. korrekte Initialisieren eines Datentyps anhand bestimmter angegebener Schlüsselwörter. Durch Modifier können Definition bestimmter Variablen verkürzt werden *[36]*
Viewer	Visualisierungskomponente eines Programms *[18]*
Workload	Aufwand für die Verarbeitung aller gleichzeitig laufenden Rechnerprozesse *[54]*

Abkürzungsverzeichnis

Abkürzung	Ausgeschriebene Form
AAA	**A**FIS (**A**mtliches **F**estpunkt**i**nformations**s**ystem), **A**LKIS (**A**mtliches **L**iegenschafts**k**ataster**i**nformations**s**ystem), **A**TKIS (**A**mtlich **T**opographisch-**K**artographisches **I**nformations-**s**ystem)
ACID	**A**tomicity, **C**onsistency, **I**solation und **D**urability
ADE	**A**pplication **D**omain **E**xtension
AdV	**A**rbeitsgemeinschaft **d**er **V**ermessungsverwaltungen der Länder der BRD
AMM	**A**utomatic **M**emory **M**anagement
API	**A**pplication **P**rogramming **I**nterface
ASSM	**A**utomatic **S**hared **M**emory **M**anagement
B-Rep	**B**oundary **Rep**resentation
BIM	**B**uilding **I**nformation **M**odel
BLOB /CLOB	**B**inary **L**arge **Ob**ject / **C**haracter **L**arge **Ob**ject
BSD	**B**erkeley **S**oftware **D**istribution
CAAD	**C**omputer-**a**ided **a**rchitectural **d**esign
CAP (Theorem)	**C**onsistency, **A**vailability, **P**artition Tolerance
COLLADA	**Colla**borative **D**esign **A**ctivity
CRS	**C**oordinate **R**eference **S**ystem
DBA	**D**atenbank**a**dministrator bzw. **A**dministration
DBMS	**D**aten**b**ank **M**anagement **S**ystem
DDL	**D**ata **D**efinition **L**anguage (SQL)
DE-9IM	**D**imensionally **E**xtended **9**-**I**ntersection **M**odel
DEM	**D**igital **E**levation **M**odel (Digitales Höhenmodell)
DGM	**D**igitales **G**elände **M**odell
DOM	**D**ocument **O**bject **M**odel
ESRI	**E**nvironmental **S**ystems **R**esearch **I**nstitute
FME	**F**eature **M**anipulation **E**ngine von SafeSoft
GDAL	**G**eospatial **D**ata **A**bstraction **L**ibrary
GDI	**G**eo**d**aten**i**nfrastruktur

GEOS	**G**eometry **E**ngine **O**pen **S**ource
GIS	**G**eographisches **I**nformations**s**ystem
GML	**G**eography **M**arkup **L**anguage
GNU	**G**NU's **n**ot **U**NIX
GPL	**G**eneral **P**ublic **L**icense
GUI	**G**raphical **U**ser **I**nterface
HTML	**H**ypertext **M**arkup **L**anguage
IFC	**I**ndustry **F**oundation **C**lasses
IGG	**I**nstitut für **G**eodäsie und **G**eoinformatstechnik der TU-Berlin
IMP	Kurzform für Hilfstabellen beim Import von Rasterdaten in der Oracle-Version der 3DCityDB
INSPIRE	**In**frastructure for **Sp**atial **I**nfo**r**mation in the **E**uropean Community
IRC	**I**nternet **R**elay **C**hat
ISO	Name der Internationalen Organisation für Normung
JAXB	**J**ava **A**rchitecture for **X**ML **B**inding
JDBC	**J**ava **D**ata**b**ase **C**onnectivity
JTS	**J**TS **T**opology **S**uite
JVM	**J**ava **V**irtual **M**achine
KML	**K**eyhole **M**arkup **L**anguage
LGPL	**L**esser **G**eneral **P**ublic **L**icence
LIDAR	**Li**ght **D**etection **a**nd **R**anging
LOD oder LoD	**L**evel **o**f **D**etail
LRS	**L**inear **R**eference **S**ystem
MB	**M**ega**b**yte
NAS	**N**ormenbasierte **A**ustausch**s**chnittstelle
OGC	**O**pen **G**eospatial **C**onsortium
OODBMS	**O**bjekt**o**rientiertes **D**aten**b**ank **M**anagement **S**ystem
ORDBMS	**O**bjekt-**r**elationales **D**aten**b**ank **M**anagement **S**ystem
OSM	**O**pen**S**treet**M**ap
PL	**P**rocedural **L**anguage

RAM	**R**andom-**A**ccess **M**emory
RDBMS	**R**elationales **D**aten**b**ank **M**anagement **S**ystem
RDT	**R**aster **D**ata **T**able
SAX	**S**imple **A**PI for **X**ML
SDE	**S**patial **D**atabase **E**ngine von ESRI
SDO	**S**patial **D**ata **O**bject
SIG 3D	**S**pecial **I**nterest **G**roup **3D**
SOA	**S**ervice-**o**rientierte **A**rchitektur
SQL	**S**tructured **Q**uery **L**anguage
SQL / MM	SQL **Mulitm**edia Application Pakages
SQL / PSM	SQL **P**ersistent **S**tored **M**odule
SRID	**S**patial **R**eference **Id**entifier
ST	**S**patial **T**emporal
TIN	**T**riangulated **I**rregular **N**etwork
UCP	**U**niversal **C**onnection **P**ool von Oracle
UML	**U**nified **M**odelling **L**anguage
URL	**U**niform **R**essource **L**ocator
URI	**U**niform **R**essource **I**dentifier
VGI	**V**olunteered **G**eographic **I**nformation
VRML	**V**irtual **R**eality **M**odeling **L**anguage
W3C	**W**orld **W**ide **W**eb **C**onsortium
W3DS	**W**eb **3D** **S**ervice
WebGL	**Web** **G**raphics **L**ibrary
WFS	**W**eb **F**eature **S**ervice
WKB	**W**ell **K**nown **B**inary
WKT / EWKT	**W**ell **K**nown **T**ext / **E**xtended **W**ell **K**nown **T**ext
WMS	**W**eb **M**ap **S**ervice
WVS	**W**eb **V**iew **S**ervice
X3D	E**x**tensible **3D**
XML	E**x**tensible **M**arkup **L**anguage

Anhang

Was erwartet den Leser?

Die Umsetzungen zur *PostGIS*-Version sind im Internet verfügbar und können von jedem Interessierten eingesehen werden. Daher war es die Absicht im Anhang über eine reine Auflistung der Skripte und Java-Klassen hinaus eine Gegenüberstellung mit der bisherigen *Oracle*-Version zu suchen, um dem Leser die Unterschiede klar vor Augen zu führen, so dass dieser den geleisteten Aufwand der Masterarbeit selbst abschätzen kann. Der Anhang umfasst alle Skripte der veröffentlichen Version und ist nicht auf die in der Masterarbeit zunächst vorgesehene Teilumsetzung beschränkt.

Nicht zuletzt werden die Ergebnisse des Performanz-Vergleichs sowie weitere Abbildungen von *Importer/Exporter*, CityGML-Viewern bzw. *Google Earth* gezeigt. Im letzten Teil des Anhangs ist ein Tutorial zum Aufsetzen der *3DCityDB* in *PostGIS* auf englisch beigefügt, das Teil des Release ist.

Anhang A

SQL und PL/pgSQL Skripte

Die folgenden Seiten präsentieren alle SQL- und PL/pgSQL-Skripte und stellen sie, so vorhanden, der *Oracle*-Version auf Doppelseiten gegenüber. Eine deutlich kürzere Zusammenfassung der wichtigsten Unterschiede ist im Release-Paket enthalten oder einzeln von der Projekt-Homepage herunterladbar [*www60*]. Anmerkungen zur Portierung sind im Stile von Kommentaren in den Skripten der *PostGIS*-Version eingebaut bzw., so kein Platz vorhanden, in den Skripten der *Oracle*-Version platziert. Punkte (. . .) sollen verdeutlichen, dass der Quellcode sich an entsprechender Stelle sehr häufig wiederholt, z.B. bei Operationen für alle Tabellen der Datenbank.

Als Orientierung wird die Ordnerstruktur der veröffentlichten Version in mehreren Tabellen aufgelistet. Die in der *Oracle*-Version enthaltenen Ordner PLANNING MANAGER und UPGRADES sind nicht im Installations-Paket enthalten.

Tab. A1: Ordnerstruktur des 3DCityDB PostGIS-Version Installations-Paketes

3dcitydb/postgis/	**Erklärung**
CREATE_DB.bat	Batchdatei, die CREATE_DB.sql aufruft (für Microsoft Windows)
DROP_DB.bat	Batchdatei, die DROP_DB.sql aufruft (für Microsoft Windows)
CREATE_DB.sh	Shellskript, das CREATE_DB.sql (UNIX/Linux & Derivate, MacOS X)
DROP_DB.sh	Shellskript, das DROP_DB.sql aufruft (UNIX/Linux & Derivate, MacOS X)
CREATE_DB.sql	führt alle SQL Skripte zum Erstellen des relationalen Schemas der 3DCityDB aus
CREATE_GEODB_PKG.sql	legt ein separates Schema geodb_pkg in der Datenbank mit Prozeduren für Funktionen des Importer/Exporter
DROP_DB.sql	löscht alle Elemente des relationalen Schema, wird aufgerufen von DROB_DB.bat / .sh
SCHEMA	beinhaltet SQL-Skripte des relationalen Schemas der 3DCityDB
PL_pgSQL/GEODB_PKG	beinhaltet SQL-Skripte die PL/pgSQL-Funktionen beinhalten
UTIL	beinhaltet Hilfsskripte für 3DCityDB

A.1. Top-Level-Dateien

Tab. A2: Unterschiede bei den Top-Level-Dateien des 3DCityDB Installations-Paketes

Datei	Oracle	PgSQL	Anmerkungen zur Portierung
CREATE_DB.bat	X	✓	verringert die Arbeit mit psql
CREATE_DB.sh	X	✓	verringert die Arbeit mit psql
CREATE_DB.sql	✓	✓	wurde mit CREATE_DB2 zusammengeführt
CREATE_DB2.sql	✓	X	s.o.
CREATE_GEODB_PKG	✓	✓	
CREATE_PLANNING_MAN.	✓	X	der Planning Manager wurde nicht portiert
DISABLE_VERSIONING	✓	X	eine Versionierung wurde nicht implementiert
DROP_DB.bat	X	✓	verringert die Arbeit mit psql
DROP_DB.sh	X	✓	verringert die Arbeit mit psql
DROP_DB.sql	✓	✓	
DROP_GEODB_PKG	✓	X	ist bereits in DROP_DB.sql enthalten
DROP_PLANNING_MAN.	✓	X	der Planning Manager wurde nicht portiert
ENABLE_VERSIONING	✓	X	eine Versionierung wurde nicht implementiert

CREATE_DB.bat (DROP_DB.sql äquivalent)

```
REM Shell script to create an instance of the 3D City Database
REM on PostgreSQL/PostGIS

REM Provide your database details here
set PGPORT=5432
set PGHOST=your_host_address
set PGUSER=your_username
set CITYDB=your_database
set PGBIN=path_to_psql

REM cd to path of the shell script
cd /d %~dp0

REM Run CREATE_DB.sql to create the 3D City Database instance
"%PGBIN%\psql" -d "%CITYDB%" -f "CREATE_DB.sql"

pause
```

CREATE_DB.sh (DROP_DB.sh äquivalent)

`#!/bin/sh` zu Beginn
`export` statt `set`
`cd "$( cd "$( dirname "$0" )" && pwd )" > /dev/null` statt `cd /d %~dp0`

Oracle

CREATE_DB.sql

```
SET SERVEROUTPUT ON
SET FEEDBACK ON
SET VER OFF

prompt
prompt
accept SRSNO NUMBER DEFAULT 81989002 PROMPT 'Please enter a valid SRID (Berlin:
81989002): '
prompt Please enter the corresponding SRSName to be used in GML exports
accept GMLSRSNAME CHAR DEFAULT
'urn:ogc:def:crs,crs:EPSG:6.12:3068,crs:EPSG:6.12:5783' prompt '  (Berlin:
urn:ogc:def:crs,crs:EPSG:6.12:3068,crs:EPSG:6.12:5783): '
accept VERSIONING CHAR DEFAULT 'no' PROMPT 'Shall versioning be enabled (yes/no,
default is no): '
prompt
prompt

VARIABLE SRID NUMBER;
VARIABLE CS_NAME VARCHAR2(256);
VARIABLE BATCHFILE VARCHAR2(50);

WHENEVER SQLERROR CONTINUE;

BEGIN
  :BATCHFILE := 'UTIL/CREATE_DB/HINT_ON_MISSING_SRS';
END;
/

BEGIN
  SELECT SRID,CS_NAME INTO :SRID,:CS_NAME FROM MDSYS.CS_SRS
  WHERE SRID=&SRSNO;

  IF (:SRID = &SRSNO) THEN
    :BATCHFILE := 'CREATE_DB2';
  END IF;
END;
/

-- Transfer the value from the bind variable to the substitution variable
column mc new_value BATCHFILE2 print
select :BATCHFILE mc from dual;

START &BATCHFILE2
```

PostgreSQL / PostGIS

CREATE_DB.sql

```
SET client_min_messages TO WARNING;
-- ansonsten relativ viel Feedback von PostgreSQL




\prompt 'Please enter a valid SRID (e.g., 3068 for DHDN/Soldner Berlin): ' SRS_NO

\prompt 'Please enter the corresponding SRSName to be used in GML exports (e.g.,
urn:ogc:def:crs,crs:EPSG::3068,crs:EPSG::5783): ' GMLSRSNAME
-- default Werte können nicht definiert werden






\set SRSNO :SRS_NO


-- bei einer fehlerhaften Eingabe soll das Skript sofort gestoppt werden
\set ON_ERROR_STOP ON
\echo

-- Die Methodik die Existenz des SRIDs in der Datenbank zu prüfen wurde abgewandelt.
-- Das Skript CHECK_SRID.sql erstellt die Funktion check_srid(integer), die prüft,
-- ob das SRID vorhanden ist und ob es auf die PostGIS-Funktionen, die der Importer/
-- Exporter aufruft, angewendet werden kann. Je nachdem welche Prüfung fehl schlägt
-- wird eine Exception geworfen, die zum Abbruch von CREATE_DB.sql (ON_ERROR_STOP).
-- Waren beide Prüfungen erfolgreich, gibt die Funktion den Text 'SRID ok' zurück,
-- der als ResultSet in der Konsole erscheint. Eine Unterteilung in die Dateien
-- CREATE_DB2.sql und HINT_ON_MISSING_SRS.sql ist daher nicht mehr notwendig. Das
-- Skript ist auf Seite 139 aufgeführt.

--// checks if the chosen SRID is provided by the spatial_ref_sys table
\i UTIL/CREATE_DB/CHECK_SRID.sql
SELECT check_srid(:SRS_NO);
```

Oracle

CREATE_DB2.sql

```
SET SERVEROUTPUT ON
SET FEEDBACK ON
SET VER OFF

VARIABLE VERSIONBATCHFILE VARCHAR2(50);

-- This script is called from CREATE_DB.sql and it
-- is required that the three substitution variables
-- &SRSNO, &GMLSRSNAME, and &VERSIONING are set properly.

--// create database srs
@@SCHEMA/TABLES/METADATA/DATABASE_SRS.sql
INSERT INTO DATABASE_SRS(SRID,GML_SRS_NAME) VALUES (&SRSNO,'&GMLSRSNAME');
COMMIT;

--// create tables
@@SCHEMA/TABLES/METADATA/OBJECTCLASS.sql
@@SCHEMA/TABLES/CORE/CITYMODEL.sql
. . .

--// create sequences
@@SCHEMA/SEQUENCES/CITYMODEL_SEQ.sql
@@SCHEMA/SEQUENCES/CITYOBJECT_SEQ.sql
. . .

--// activate constraints
@@SCHEMA/CONSTRAINTS/CONSTRAINTS.sql

--// BUILD INDEXES
@@SCHEMA/INDEXES/SIMPLE_INDEX.sql
@@SCHEMA/INDEXES/SPATIAL_INDEX.sql

@@UTIL/CREATE_DB/OBJECTCLASS_INSTANCES.sql
@@UTIL/CREATE_DB/IMPORT_PROCEDURES.sql
@@UTIL/CREATE_DB/DUMMY_IMPORT.sql

--// (possibly) activate versioning
BEGIN
  :VERSIONBATCHFILE := 'UTIL/CREATE_DB/DO_NOTHING.sql';
END;
/
BEGIN
  IF ('&VERSIONING'='yes' OR '&VERSIONING'='YES' OR '&VERSIONING'='y' OR
      '&VERSIONING'='Y') THEN
    :VERSIONBATCHFILE := 'ENABLE_VERSIONING.sql';
  END IF;
END;
/
```

PostgreSQL / PostGIS

CREATE_DB.sql (Fortführung)

```
\i SCHEMA/TABLES/METADATA/DATABASE_SRS.sql
INSERT INTO DATABASE_SRS(SRID,GML_SRS_NAME) VALUES (:SRS_NO,:'GMLSRSNAME');


--// create TABLES
\i SCHEMA/TABLES/METADATA/OBJECTCLASS.sql
\i SCHEMA/TABLES/CORE/CITYMODEL.sql
. . .


-- Sequenzen werden implizit mit dem Datentyp SERIAL erstellt. Alle entsprechenden
-- Dateien der Oracle-Version entfallen damit.


--// activate constraints
\i SCHEMA/CONSTRAINTS/CONSTRAINTS.sql

--// build INDEXES
\i SCHEMA/INDEXES/SIMPLE_INDEX.sql
\i SCHEMA/INDEXES/SPATIAL_INDEX.sql

--// fill table OBJECTCLASS
\i UTIL/CREATE_DB/OBJECTCLASS_INSTANCES.sql

-- Die Dateien IMPORT_PROCEDURES.sql und DUMMY_IMPORT.sql dienten dem bisherigen
-- Rasterdaten-Management. Da es, wie in der Arbeit erläutert, stark vereinfacht
-- wurde, fielen diese Dateien weg.

-- Alle Skripte und Zeilen, welche die Versionierung betreffen, sind nicht portiert
-- worden.
```

Oracle

CREATE_DB2.sql (Fortführung)

```
-- Transfer the value from the bind variable to the substitution variable
column mc2 new_value VERSIONBATCHFILE2 print
select :VERSIONBATCHFILE mc2 from dual;
@@&VERSIONBATCHFILE2

--// DML TRIGGER FOR RASTER TABLES
@@SCHEMA/TRIGGER/RASTER_ORTHOPHOTO/TRIGGER.sql;

--// CREATE TABLES & PROCEDURES OF THE PLANNINGMANAGER
@@PL_SQL/MOSAIC/MOSAIC.sql;
@@CREATE_PLANNING_MANAGER.sql

--// geodb packages
@@CREATE_GEODB_PKG.sql

SHOW ERRORS;
COMMIT;

SELECT 'DB creation complete!' as message from DUAL;
```

CREATE_GEODB_PKG.sql

```
SELECT 'Creating packages ''geodb_util'', ''geodb_idx'', ''geodb_stat'',
''geodb_delete_by_lineage'', ''geodb_delete'', and corresponding types' as message
from DUAL;

@@PL_SQL/GEODB_PKG/UTIL/UTIL.sql;
@@PL_SQL/GEODB_PKG/INDEX/IDX.sql;
@@PL_SQL/GEODB_PKG/STATISTICS/STAT.sql;
@@PL_SQL/GEODB_PKG/DELETE/DELETE.sql;
@@PL_SQL/GEODB_PKG/DELETE/DELETE_BY_LINEAGE;

SELECT 'Packages ''geodb_util'', ''geodb_idx'', ''geodb_stat'',
''geodb_delete_by_lineage'', and ''geodb_delete'' created' as message from DUAL;

SELECT 'Creating matching tool packages ''geodb_match'', ''geodb_merge'', and
corresponding types' as message from DUAL;

@@PL_SQL/GEODB_PKG/MATCHING/MATCH.sql;
@@PL_SQL/GEODB_PKG/MATCHING/MERGE.sql;

SELECT 'Packages ''geodb_match'', and ''geodb_merge'' created' as message from
DUAL;
```

PostgreSQL / PostGIS

CREATE_DB.sql (Fortführung)

```
-- Die TRIGGER dienten dem bisherigen Rasterdaten-Management. Da es, wie in der
-- Arbeit erläutert, stark vereinfacht wurde, entfiel die Datei.

-- Da der Planning Manager nicht portiert wurde, entfallen die Dateien



--// create GEODB_PKG (additional schema with PL_pgSQL-Functions)
\i CREATE_GEODB_PKG.sql

-- PostgreSQL ist standardmäßig auf Auto-Commit eingestellt, daher kein explizites
-- COMMIT;
\echo
\echo '3DCityDB creation complete!'
```

CREATE_GEODB_PKG.sql

```
--// create GEODB_PKG schema
CREATE SCHEMA geodb_pkg;

--// call PL/pgSQL-Scripts to add GEODB_PKG-Functions
\i PL_pgSQL/GEODB_PKG/UTIL/UTIL.sql
\i PL_pgSQL/GEODB_PKG/INDEX/IDX.sql
\i PL_pgSQL/GEODB_PKG/STATISTICS/STAT.sql
\i PL_pgSQL/GEODB_PKG/DELETE/DELETE.sql
\i PL_pgSQL/GEODB_PKG/DELETE/DELETE_BY_LINEAGE.sql







\i PL_pgSQL/GEODB_PKG/MATCHING/MATCH.sql
\i PL_pgSQL/GEODB_PKG/MATCHING/MERGE.sql
```

Oracle

DROP_DB.sql

```
--//DROP FOREIGN KEYS

ALTER TABLE ADDRESS_TO_BUILDING DROP CONSTRAINT "ADDRESS_TO_BUILDING_FK";
. . .

--//DROP TABLES AND SEQUENCES

DROP TABLE ADDRESS CASCADE CONSTRAINTS;
DROP SEQUENCE ADDRESS_SEQ;
. . .

@@DROP_PLANNING_MANAGER.sql

@@DROP_GEODB_PKG.sql

PURGE RECYCLEBIN;
```

PostgreSQL / PostGIS

DROP_DB.sql

```
--//DROP FOREIGN KEYS

ALTER TABLE ADDRESS_TO_BUILDING DROP CONSTRAINT ADDRESS_TO_BUILDING_FK;
. . .

--//DROP TABLES

DROP TABLE ADDRESS CASCADE;
-- Das explizite Löschen der Sequenzen ist nicht notwendig.
. . .


--//DROP SCHEMA
DROP SCHEMA GEODB_PKG CASCADE;
-- Ersetzt die Datei DROP_GEODB_PKG.sql

\echo
\echo '3DCityDB schema successfully removed!'
```

A.2. Schema-Ordner

Tab. A3a: Der Inhalt vom Schema-Ordner im Überblick

3dcitydb/postgis/SCHEMA				Oracle	PgSQL	Anmerkungen
			CONSTRAINTS			
			CONSTRAINTS.sql	✓	✓	
			INDEXES			
			SIMPLE_INDEX.sql	✓	✓	
			SPATIAL_INDEX.sql	✓	✓	
			SEQUENCES	✓	**X**	keine Portierung notwendig
			TABLES			
			APPEARANCE	✓	✓	
			APPEAR_TO_SURFACE_DATA.sql	✓	✓	
			APPEARANCE.sql	✓	✓	
			SURFACE_DATA.sql	✓	✓	
			TEXTUREPARAM.sql	✓	✓	
			BUILDING	✓	✓	
			ADDRESS.sql	✓	✓	
			ADDRESS_TO_BUILDING.sql	✓	✓	
			BUILDING_FURNITURE.sql	✓	✓	
			BUILDING_INSTALLATION.sql	✓	✓	
			OPENING.sql	✓	✓	
			OPENING_TO_THEM_SURFACE.sql	✓	✓	
			ROOM.sql	✓	✓	
			THEMATIC_SURFACE.sql	✓	✓	
			CITY_FURNITURE			
			CITY_FURNITURE.sql	✓	✓	
			CITYOBJECTGROUP			
			CITYOBJECTGROUP.sql	✓	✓	
			GROUP_TO_CITYOBJECTGROUP.sql	✓	✓	
			CORE			
			CITYMODEL.sql	✓	✓	
			CITYOBJECT.sql	✓	✓	
			CITYOBJECT_MEMBER.sql	✓	✓	
			EXTERNAL_REFERENCE.sql	✓	✓	
			GENERALIZATION.sql	✓	✓	
			IMPLICIT_GEOMETRY.sql	✓	✓	
			GENERICS			
			CITYOBJECT_GENERICATTRIB	✓	✓	
			GENERIC_CITYOBJECT	✓	✓	

Tab. A3b: Fortsetzung

Datei				Oracle	PgSQL	Anmerkungen
	📁		GEOMETRY	✓	✓	
		📄	SURFACE_GEOMETRY.sql	✓	✓	
	📁		LANDUSE	✓	✓	
		📄	LAND_USE.sql	✓	✓	
	📁		METADATA	✓	✓	
		📄	DATABASE_SRS	✓	✓	
		📄	OBJECTCLASS	✓	✓	
	📁		ORTHOPHOTO			
		📄	ORTHOPHOTO.sql	✓	✓	
		📄	ORTHOPHOTO_IMP.sql	✓	X	keine Portierung notwendig
		📄	ORTHOPHOTO_RDT.sql	✓	X	keine Portierung notwendig
		📄	ORTHOPHOTO_RDT_IMP.sql	✓	X	keine Portierung notwendig
	📁		RELIEF			
		📄	BREAKLINE_RELIEF.sql	✓	✓	
		📄	MASSPOINT_RELIEF.sql	✓	✓	
		📄	RASTER_RELIEF.sql	✓	✓	
		📄	RASTER_RELIEF_IMP.sql	✓	X	keine Portierung notwendig
		📄	RASTER_RELIEF_IMP_RDT.sql	✓	X	keine Portierung notwendig
		📄	RASTER_RELIEF_RDT.sql	✓	X	keine Portierung notwendig
		📄	RELIEF.sql	✓	✓	
		📄	RELIEF_COMPONENT.sql	✓	✓	
		📄	RELIEF_FEAT_TO_REL_COMP.sql	✓	✓	
		📄	RELIEF_FEATURE.sql	✓	✓	
		📄	TIN_RELIEF.sql	✓	✓	
	📁		TRANSPORTATION			
		📄	TRAFFIC_AREA.sql	✓	✓	
		📄	TRANSPORTATION_COMPLEX.sql	✓	✓	
	📁		VEGETATION			
		📄	PLANT_COVER.sql	✓	✓	
		📄	SOLITARY_VEGETAT_OBJECT.sql	✓	✓	
	📁		WATERBODY			
		📄	WATERBOD_TO_WATERBND_SRF.sql	✓	✓	
		📄	WATERBODY.sql	✓	✓	
		📄	WATERBOUNDARY_SURFACE.sql	✓	✓	
	📁		**TRIGGER**	✓	X	keine Portierung notwendig

Oracle

CONSTRAINTS.sql

```
--//ADDRESS_TO_BUILDING CONSTRAINTS

ALTER TABLE ADDRESS_TO_BUILDING
ADD CONSTRAINT ADDRESS_TO_BUILDING_FK FOREIGN KEY (BUILDING_ID)
REFERENCES BUILDING (ID) ENABLE;
. . .
```

SIMPLE_INDEX.sql

```
CREATE INDEX ADDRESS_TO_BUILDING_FKX ON ADDRESS_TO_BUILDING (BUILDING_ID);
. . .
```

SPATIAL_INDEX.sql

```
--// ENTRIES INTO USER_SDO_GEOM_METADATA WITH BOUNDING VOLUME WITH 10000km EXTENT
-- FOR X,Y AND 11km FOR Z

SET SERVEROUTPUT ON
SET FEEDBACK ON

ALTER SESSION set NLS_TERRITORY='AMERICA';
ALTER SESSION set NLS_LANGUAGE='AMERICAN';

-- Fetch SRID from the the table DATABASE_SRS
VARIABLE SRID NUMBER;
BEGIN
  SELECT SRID INTO :SRID FROM DATABASE_SRS;
END;
/

-- Transfer the value from the bind variable :SRID to the substitution variable
&SRSNO
column mc new_value SRSNO print
select :SRID mc from dual;

prompt Used SRID for spatial indexes: &SRSNO

DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME='CITYOBJECT' AND
COLUMN_NAME='ENVELOPE';
INSERT INTO USER_SDO_GEOM_METADATA (TABLE_NAME, COLUMN_NAME, DIMINFO, SRID)
  VALUES ('CITYOBJECT', 'ENVELOPE',
    MDSYS.SDO_DIM_ARRAY (
       MDSYS.SDO_DIM_ELEMENT('X', 0.000, 10000000.000, 0.0005),
       MDSYS.SDO_DIM_ELEMENT('Y', 0.000, 10000000.000, 0.0005),
       MDSYS.SDO_DIM_ELEMENT('Z', -1000, 10000, 0.0005)),  &SRSNO);
. . .
```

PostgreSQL / PostGIS

CONSTRAINTS.sql

```
--//ADDRESS_TO_BUILDING CONSTRAINTS

ALTER TABLE ADDRESS_TO_BUILDING
ADD CONSTRAINT ADDRESS_TO_BUILDING_FK FOREIGN KEY (BUILDING_ID)
REFERENCES BUILDING (ID)
ON UPDATE CASCADE ON DELETE RESTRICT;
. . .
```

SIMPLE_INDEX.sql

```
CREATE INDEX ADDRESS_TO_BUILDING_FKX ON ADDRESS_TO_BUILDING (BUILDING_ID);
. . .
```

SPATIAL_INDEX.sql

```
-- In Oracle müssen die Metadaten für die Geometriespalten explizit angeben werden
-- Sie werden u.a. für räumliche Indizes benötigt.
-- In PostGIS werden die Metadaten implizit beim Erstellen der Tabelle in dem View
-- geometry_columns angelegt (siehe entsprechende Skripte)
```

Oracle

SPATIAL_INDEX.sql (Fortführung)

```
DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME='SURFACE_DATA' AND
COLUMN_NAME='GT_REFERENCE_POINT';
INSERT INTO USER_SDO_GEOM_METADATA (TABLE_NAME, COLUMN_NAME, DIMINFO, SRID)
  VALUES ('SURFACE_DATA', 'GT_REFERENCE_POINT',
    MDSYS.SDO_DIM_ARRAY (
       MDSYS.SDO_DIM_ELEMENT('X', 0.000, 10000000.000, 0.0005),
       MDSYS.SDO_DIM_ELEMENT('Y', 0.000, 10000000.000, 0.0005)), &SRSNO);
. . .

--// CREATE INDEX STATEMENTS

-- CITYOBJECT_ENVELOPE
CREATE INDEX CITYOBJECT_SPX on CITYOBJECT(ENVELOPE)
  INDEXTYPE is MDSYS.SPATIAL_INDEX;
. . .

-- SURFACE_DATA
CREATE INDEX SURFACE_DATA_SPX on SURFACE_DATA(GT_REFERENCE_POINT)
  INDEXTYPE is MDSYS.SPATIAL_INDEX;
. . .
```

APPEAR_TO_SURFACE_DATA.sql

```
CREATE TABLE APPEAR_TO_SURFACE_DATA (
  SURFACE_DATA_ID     NUMBER NOT NULL,
  APPEARANCE_ID       NUMBER NOT NULL
);

ALTER TABLE APPEAR_TO_SURFACE_DATA
ADD CONSTRAINT APPEAR_TO_SURFACE_DATA_PK PRIMARY KEY (
  SURFACE_DATA_ID,
  APPEARANCE_ID
)ENABLE;
```

APPEARANCE.sql

```
CREATE TABLE APPEARANCE (
  ID                 NUMBER NOT NULL,
  GMLID              VARCHAR2(256),
  GMLID_CODESPACE    VARCHAR2(1000),
  NAME               VARCHAR2(1000),
  NAME_CODESPACE     VARCHAR2(4000),
  DESCRIPTION        VARCHAR2(4000),
  THEME              VARCHAR2(256),
  CITYMODEL_ID       NUMBER,
  CITYOBJECT_ID      NUMBER
); . . .
```

PostgreSQL / PostGIS

SPATIAL_INDEX.sql (Fortführung)

```
-- CITYOBJECT SPATIAL INDEX*******************************************************
CREATE INDEX CITYOBJECT_SPX ON CITYOBJECT USING
  GIST ( ENVELOPE gist_geometry_ops_nd ); -- mehrdimensionaler Index
. . .

-- SURFACE_DATA SPATIAL INDEX*****************************************************
CREATE INDEX SURFACE_DATA_SPX ON SURFACE_DATA USING
  GIST ( GT_REFERENCE_POINT );
. . .
```

APPEAR_TO_SURFACE_DATA.sql

```
CREATE TABLE APPEAR_TO_SURFACE_DATA (
  SURFACE_DATA_ID     INTEGER NOT NULL,
  APPEARANCE_ID       INTEGRE NOT NULL
);

ALTER TABLE APPEAR_TO_SURFACE_DATA
ADD CONSTRAINT APPEAR_TO_SURFACE_DATA_PK PRIMARY KEY (
  SURFACE_DATA_ID,
  APPEARANCE_ID
); -- kein ENABLE / DISABLE in PostgreSQL, es gibt aber „disable trigger all"
```

APPEARANCE.sql

```
CREATE TABLE APPEARANCE (
  ID                  SERIAL NOT NULL,
  GMLID               VARCHAR(256),
  GMLID_CODESPACE     VARCHAR(1000),
  NAME                VARCHAR(1000),
  NAME_CODESPACE      VARCHAR(4000),
  DESCRIPTION         VARCHAR(4000),
  THEME               VARCHAR(256),
  CITYMODEL_ID        INTEGER,
  CITYOBJECT_ID       INTEGER
); . . .
```

Oracle

SURFACE_DATA.sql

```
CREATE TABLE SURFACE_DATA (
  ID                      NUMBER NOT NULL,
  GMLID                   VARCHAR2(256),
  GMLID_CODESPACE         VARCHAR2(1000),
  NAME                    VARCHAR2(1000),
  NAME_CODESPACE          VARCHAR2(4000),
  DESCRIPTION             VARCHAR2(4000),
  IS_FRONT                NUMBER(1, 0),
  TYPE                    VARCHAR2(30),
  X3D_SHININESS           BINARY_DOUBLE,
  X3D_TRANSPARENCY        BINARY_DOUBLE,
  X3D_AMBIENT_INTENSITY   BINARY_DOUBLE,
  X3D_SPECULAR_COLOR      VARCHAR2(256),
  X3D_DIFFUSE_COLOR       VARCHAR2(256),
  X3D_EMISSIVE_COLOR      VARCHAR2(256),
  X3D_IS_SMOOTH           NUMBER(1, 0),
  TEX_IMAGE_URI           VARCHAR2(4000),
  TEX_IMAGE               ORDSYS.ORDIMAGE,
  TEX_MIME_TYPE           VARCHAR2(256),
  TEX_TEXTURE_TYPE        VARCHAR2(256),
  TEX_WRAP_MODE           VARCHAR2(256),
  TEX_BORDER_COLOR        VARCHAR2(256),
  GT_PREFER_WORLDFILE     NUMBER(1, 0),
  GT_ORIENTATION          VARCHAR2(256),
  GT_REFERENCE_POINT      MDSYS.SDO_GEOMETRY
); . . .
```

TEXTUREPARAM.sql

```
CREATE TABLE TEXTUREPARAM
(
  SURFACE_GEOMETRY_ID          NUMBER NOT NULL,
  IS_TEXTURE_PARAMETRIZATION   NUMBER(1, 0),
  WORLD_TO_TEXTURE             VARCHAR2(1000),
  TEXTURE_COORDINATES          VARCHAR2(4000),
  SURFACE_DATA_ID              NUMBER NOT NULL
); . . .
```

PostgreSQL / PostGIS

SURFACE_DATA.sql

```
CREATE TABLE SURFACE_DATA (
  ID                       SERIAL NOT NULL,
  GMLID                    VARCHAR(256),
  GMLID_CODESPACE          VARCHAR(1000),
  NAME                     VARCHAR(1000),
  NAME_CODESPACE           VARCHAR(4000),
  DESCRIPTION              VARCHAR(4000),
  IS_FRONT                 NUMERIC(1, 0),
  TYPE                     VARCHAR(30),
  X3D_SHININESS            DOUBLE PRECISION,
  X3D_TRANSPARENCY         DOUBLE PRECISION,
  X3D_AMBIENT_INTENSITY    DOUBLE PRECISION,
  X3D_SPECULAR_COLOR       VARCHAR(256),
  X3D_DIFFUSE_COLOR        VARCHAR(256),
  X3D_EMISSIVE_COLOR       VARCHAR(256),
  X3D_IS_SMOOTH            NUMERIC(1, 0),
  TEX_IMAGE_URI            VARCHAR(4000),
  TEX_IMAGE                BYTEA,
  TEX_MIME_TYPE            VARCHAR(256),
  TEX_TEXTURE_TYPE         VARCHAR(256),
  TEX_WRAP_MODE            VARCHAR(256),
  TEX_BORDER_COLOR         VARCHAR(256),
  GT_PREFER_WORLDFILE      NUMERIC(1, 0),
  GT_ORIENTATION           VARCHAR(256),
  GT_REFERENCE_POINT       GEOMETRY(Point,:SRSNO)
); . . .
```

TEXTUREPARAM.sql

```
CREATE TABLE TEXTUREPARAM
(
  SURFACE_GEOMETRY_ID           SERIAL NOT NULL,
  IS_TEXTURE_PARAMETRIZATION    NUMERIC(1, 0),
  WORLD_TO_TEXTURE              VARCHAR(1000),
  TEXTURE_COORDINATES           VARCHAR(4000),
  SURFACE_DATA_ID               INTEGER NOT NULL
); . . .
```

Oracle

ADDRESS.sql

```
CREATE TABLE ADDRESS (
  ID              NUMBER NOT NULL,
  STREET          VARCHAR2(1000),
  HOUSE_NUMBER    VARCHAR2(256),
  PO_BOX          VARCHAR2(256),
  ZIP_CODE        VARCHAR2(256),
  CITY            VARCHAR2(256),
  STATE           VARCHAR2(256),
  COUNTRY         VARCHAR2(256),
  MULTI_POINT     MDSYS.SDO_GEOMETRY,
  XAL_SOURCE      CLOB
); . . .
```

ADDRESS_TO_BUILDING.sql

```
CREATE TABLE ADDRESS_TO_BUILDING
(
  BUILDING_ID    NUMBER NOT NULL,
  ADDRESS_ID     NUMBER NOT NULL
); . . .
```

ADDRESS_TO_BUILDING.sql

```
CREATE TABLE BUILDING (
  ID                            NUMBER NOT NULL,
  NAME                          VARCHAR2(1000),
  NAME_CODESPACE                VARCHAR2(4000),
  BUILDING_PARENT_ID            NUMBER,
  BUILDING_ROOT_ID              NUMBER,
  DESCRIPTION                   VARCHAR2(4000),
  CLASS                         VARCHAR2(256),
  FUNCTION                      VARCHAR2(1000),
  USAGE                         VARCHAR2(1000),
  YEAR_OF_CONSTRUCTION          DATE,
  YEAR_OF_DEMOLITION            DATE,
  ROOF_TYPE                     VARCHAR2(256),
  MEASURED_HEIGHT               BINARY_DOUBLE,
  STOREYS_ABOVE_GROUND          NUMBER(8),
  STOREYS_BELOW_GROUND          NUMBER(8),
  STOREY_HEIGHTS_ABOVE_GROUND   VARCHAR2(4000),
  STOREY_HEIGHTS_BELOW_GROUND   VARCHAR2(4000),
  LOD1_TERRAIN_INTERSECTION     MDSYS.SDO_GEOMETRY, . . . -- auch für LOD 2,3,4
  LOD2_MULTI_CURVE              MDSYS.SDO_GEOMETRY, . . . -- auch für LOD 3,4
  LOD4_GEOMETRY_ID              NUMBER                . . . -- auch für LOD 1,2,3
); . . .
```

PostgreSQL / PostGIS

ADDRESS.sql

```
CREATE TABLE ADDRESS (
  ID              SERIAL NOT NULL,
  STREET          VARCHAR(1000),
  HOUSE_NUMBER    VARCHAR(256),
  PO_BOX          VARCHAR(256),
  ZIP_CODE        VARCHAR(256),
  CITY            VARCHAR(256),
  STATE           VARCHAR(256),
  COUNTRY         VARCHAR(256),
  MULTI_POINT     GEOMETRY(MultiPointZ,:SRSNO),
  XAL_SOURCE      TEXT
); . . .
```

ADDRESS_TO_BUILDING.sql

```
CREATE TABLE ADDRESS_TO_BUILDING
(
  BUILDING_ID    INTEGER NOT NULL,
  ADDRESS_ID     INTEGER NOT NULL
); . . .
```

ADDRESS_TO_BUILDING.sql

```
CREATE TABLE BUILDING (
  ID                             SERIAL NOT NULL,
  NAME                           VARCHAR(1000),
  NAME_CODESPACE                 VARCHAR(4000),
  BUILDING_PARENT_ID             INTEGER,
  BUILDING_ROOT_ID               INTEGER,
  DESCRIPTION                    VARCHAR(4000),
  CLASS                          VARCHAR(256),
  FUNCTION                       VARCHAR(1000),
  USAGE                          VARCHAR(1000),
  YEAR_OF_CONSTRUCTION           DATE,
  YEAR_OF_DEMOLITION             DATE,
  ROOF_TYPE                      VARCHAR(256),
  MEASURED_HEIGHT                DOUBLE PRECISION,
  STOREYS_ABOVE_GROUND           NUMERIC(8),
  STOREYS_BELOW_GROUND           NUMERIC(8),
  STOREY_HEIGHTS_ABOVE_GROUND    VARCHAR(4000),
  STOREY_HEIGHTS_BELOW_GROUND    VARCHAR(4000),
  LOD1_TERRAIN_INTERSECTION      GEOMETRY(GeometryZ,:SRSNO),    . . . -- siehe links
  LOD2_MULTI_CURVE               GEOMETRY(MultiCurveZ,:SRSNO), . . .
  LOD4_GEOMETRY_ID               INTEGER
); . . .
```

Oracle

BUILDING_FURNITURE.sql

```
CREATE TABLE BUILDING_FURNITURE (
  ID                          NUMBER NOT NULL,
  NAME                        VARCHAR2(1000),
  NAME_CODESPACE              VARCHAR2(4000),
  DESCRIPTION                 VARCHAR2(4000),
  CLASS                       VARCHAR2(256),
  FUNCTION                    VARCHAR2(1000),
  USAGE                       VARCHAR2(1000),
  ROOM_ID                     NUMBER NOT NULL,
  LOD4_GEOMETRY_ID            NUMBER,
  LOD4_IMPLICIT_REP_ID        NUMBER,
  LOD4_IMPLICIT_REF_POINT     MDSYS.SDO_GEOMETRY,
  LOD4_IMPLICIT_TRANSFORMATION  VARCHAR2(1000)
); . . .
```

BUILDING_INSTALLATION.sql

```
CREATE TABLE BUILDING_INSTALLATION (
  ID                          NUMBER NOT NULL,
  IS_EXTERNAL                 NUMBER(1, 0),
  NAME                        VARCHAR2(1000),
  NAME_CODESPACE              VARCHAR2(4000),
  DESCRIPTION                 VARCHAR2(4000),
  CLASS                       VARCHAR2(256),
  FUNCTION                    VARCHAR2(1000),
  USAGE                       VARCHAR2(1000),
  BUILDING_ID                 NUMBER,
  ROOM_ID                     NUMBER,
  LOD2_GEOMETRY_ID            NUMBER,              . . . -- auch für LOD 3,4
); . . .
```

OPENING.sql

```
CREATE TABLE OPENING (
  ID                          NUMBER NOT NULL,
  NAME                        VARCHAR2(1000),
  NAME_CODESPACE              VARCHAR2(4000),
  DESCRIPTION                 VARCHAR2(4000),
  TYPE                        VARCHAR2(256),
  ADDRESS_ID                  NUMBER,
  LOD3_MULTI_SURFACE_ID       NUMBER,              . . . -- auch für LOD 4
); . . .
```

PostgreSQL / PostGIS

BUILDING_FURNITURE.sql

```
CREATE TABLE BUILDING_FURNITURE (
  ID                            SERIAL NOT NULL,
  NAME                          VARCHAR(1000),
  NAME_CODESPACE                VARCHAR(4000),
  DESCRIPTION                   VARCHAR(4000),
  CLASS                         VARCHAR(256),
  FUNCTION                      VARCHAR(1000),
  USAGE                         VARCHAR(1000),
  ROOM_ID                       INTEGER NOT NULL,
  LOD4_GEOMETRY_ID              INTEGER,
  LOD4_IMPLICIT_REP_ID          INTEGER,
  LOD4_IMPLICIT_REF_POINT       GEOMETRY(PointZ,:SRSNO,
  LOD4_IMPLICIT_TRANSFORMATION  VARCHAR(1000)
); . . .
```

BUILDING_INSTALLATION.sql

```
CREATE TABLE BUILDING_INSTALLATION (
  ID                            SERIAL NOT NULL,
  IS_EXTERNAL                   NUMERIC(1, 0),
  NAME                          VARCHAR(1000),
  NAME_CODESPACE                VARCHAR(4000),
  DESCRIPTION                   VARCHAR(4000),
  CLASS                         VARCHAR(256),
  FUNCTION                      VARCHAR(1000),
  USAGE                         VARCHAR(1000),
  BUILDING_ID                   INTEGER,
  ROOM_ID                       INTEGER,
  LOD2_GEOMETRY_ID              INTEGER,           . . . -- siehe links
); . . .
```

OPENING.sql

```
CREATE TABLE OPENING (
  ID                            SERIAL NOT NULL,
  NAME                          VARCHAR(1000),
  NAME_CODESPACE                VARCHAR(4000),
  DESCRIPTION                   VARCHAR(4000),
  TYPE                          VARCHAR(256),
  ADDRESS_ID                    INTEGER,
  LOD3_MULTI_SURFACE_ID         INTEGER,           . . . -- siehe links
); . . .
```

Oracle

OPENING_TO_THEM_SURFACE.sql

```
CREATE TABLE OPENING_TO_THEM_SURFACE
(
  OPENING_ID                   NUMBER NOT NULL,
  THEMATIC_SURFACE_ID          NUMBER NOT NULL
); . . .
```

ROOM.sql

```
CREATE TABLE ROOM (
  ID                           NUMBER NOT NULL,
  NAME                         VARCHAR2(1000),
  NAME_CODESPACE               VARCHAR2(4000),
  DESCRIPTION                  VARCHAR2(4000),
  CLASS                        VARCHAR2(256),
  FUNCTION                     VARCHAR2(1000),
  USAGE                        VARCHAR2(1000),
  BUILDING_ID                  NUMBER NOT NULL,
  LOD4_GEOMETRY_ID             NUMBER
); . . .
```

THEMATIC_SURFACE.sql

```
CREATE TABLE THEMATIC_SURFACE (
  ID                           NUMBER NOT NULL,
  NAME                         VARCHAR2(1000),
  NAME_CODESPACE               VARCHAR2(4000),
  DESCRIPTION                  VARCHAR2(4000),
  TYPE                         VARCHAR2(256),
  LOD2_MULTI_SURFACE_ID        NUMBER,              . . . -- auch für LOD 3,4
); . . .
```

CITY_FURNITURE.sql

```
CREATE TABLE CITY_FURNITURE (
  ID                           NUMBER NOT NULL,
  NAME                         VARCHAR2(1000),
  NAME_CODESPACE               VARCHAR2(4000),
  DESCRIPTION                  VARCHAR2(4000),
  CLASS                        VARCHAR2(256),
  FUNCTION                     VARCHAR2(1000),
  LOD1_TERRAIN_INTERSECTION    MDSYS.SDO_GEOMETRY, . . . -- auch für LOD 2,3,4
  LOD1_GEOMETRY_ID             NUMBER,              . . . -- auch für LOD 2,3,4
  LOD1_IMPLICIT_REP_ID         NUMBER,              . . . -- auch für LOD 2,3,4
  LOD1_IMPLICIT_REF_POINT      MDSYS.SDO_GEOMETRY, . . . -- auch für LOD 2,3,4
  LOD4_IMPLICIT_TRANSFORMATION VARCHAR2(1000)       . . . -- auch für LOD 1,2,3
);
```

PostgreSQL / PostGIS

OPENING_TO_THEM_SURFACE.sql

```
CREATE TABLE OPENING_TO_THEM_SURFACE
(
  OPENING_ID                      INTEGER NOT NULL,
  THEMATIC_SURFACE_ID             INTEGER NOT NULL
); . . .
```

ROOM.sql

```
CREATE TABLE ROOM (
  ID                              SERIAL NOT NULL,
  NAME                            VARCHAR(1000),
  NAME_CODESPACE                  VARCHAR(4000),
  DESCRIPTION                     VARCHAR(4000),
  CLASS                           VARCHAR(256),
  FUNCTION                        VARCHAR(1000),
  USAGE                           VARCHAR(1000),
  BUILDING_ID                     INTEGER NOT NULL,
  LOD4_GEOMETRY_ID                INTEGER
); . . .
```

THEMATIC_SURFACE.sql

```
CREATE TABLE THEMATIC_SURFACE (
  ID                              SERIAL NOT NULL,
  NAME                            VARCHAR(1000),
  NAME_CODESPACE                  VARCHAR(4000),
  DESCRIPTION                     VARCHAR(4000),
  TYPE                            VARCHAR(256),
  LOD2_MULTI_SURFACE_ID           INTEGER,                    . . . -- siehe links
); . . .
```

CITY_FURNITURE.sql

```
CREATE TABLE CITY_FURNITURE (
  ID                              SERIAL NOT NULL,
  NAME                            VARCHAR(1000),
  NAME_CODESPACE                  VARCHAR(4000),
  DESCRIPTION                     VARCHAR(4000),
  CLASS                           VARCHAR(256),
  FUNCTION                        VARCHAR(1000),
  LOD1_TERRAIN_INTERSECTION       GEOMETRY(GeometryZ,:SRSNO), . . . -- siehe links
  LOD1_GEOMETRY_ID                INTEGER,                    . . .
  LOD1_IMPLICIT_REP_ID            INTEGER,                    . . .
  LOD1_IMPLICIT_REF_POINT         GEOMETRY(PointZ,:SRSNO),    . . .
  LOD4_IMPLICIT_TRANSFORMATION    VARCHAR(1000)               . . .
);
```

Oracle

CITYOBJECTGROUP.sql

```
CREATE TABLE CITYOBJECTGROUP (
  ID                        NUMBER NOT NULL,
  NAME                      VARCHAR2(1000),
  NAME_CODESPACE            VARCHAR2(4000),
  DESCRIPTION               VARCHAR2(4000),
  CLASS                     VARCHAR2(256),
  FUNCTION                  VARCHAR2(1000),
  USAGE                     VARCHAR2(1000),
  GEOMETRY                  MDSYS.SDO_GEOMETRY,
  SURFACE_GEOMETRY_ID       NUMBER,
  PARENT_CITYOBJECT_ID      NUMBER
); . . .
```

GROUP_TO_CITYOBJECT.sql

```
CREATE TABLE GROUP_TO_CITYOBJECT
(
  CITYOBJECT_ID             NUMBER NOT NULL,
  CITYOBJECTGROUP_ID        NUMBER NOT NULL,
  ROLE                      VARCHAR2(256)
); . . .
```

CITYMODEL.sql

```
CREATE TABLE CITYMODEL (
  ID                        NUMBER NOT NULL,
  GMLID                     VARCHAR2(256),
  GMLID_CODESPACE           VARCHAR2(1000),
  NAME                      VARCHAR2(1000),
  NAME_CODESPACE            VARCHAR2(4000),
  DESCRIPTION               VARCHAR2(4000),
  ENVELOPE                  MDSYS.SDO_GEOMETRY,
  CREATION_DATE             DATE,
  TERMINATION_DATE          DATE,
  LAST_MODIFICATION_DATE    DATE,
  UPDATING_PERSON           VARCHAR2(256),
  REASON_FOR_UPDATE         VARCHAR2(4000),
  LINEAGE                   VARCHAR2(256)
); . . .
```

CITYOBJECT_MEMBER.sql

```
CREATE TABLE CITYOBJECT_MEMBER (
  CITYMODEL_ID             NUMBER NOT NULL,
  CITYOBJECT_ID            NUMBER NOT NULL
); . . .
```

PostgreSQL / PostGIS

CITYOBJECTGROUP.sql

```
CREATE TABLE CITYOBJECTGROUP (
  ID                         SERIAL NOT NULL,
  NAME                       VARCHAR(1000),
  NAME_CODESPACE             VARCHAR(4000),
  DESCRIPTION                VARCHAR(4000),
  CLASS                      VARCHAR(256),
  FUNCTION                   VARCHAR(1000),
  USAGE                      VARCHAR(1000),
  GEOMETRY                   GEOMETRY(PolygonZ,:SRSNO),
  SURFACE_GEOMETRY_ID        INTEGER,
  PARENT_CITYOBJECT_ID       INTEGER
); . . .
```

GROUP_TO_CITYOBJECT.sql

```
CREATE TABLE GROUP_TO_CITYOBJECT
(
  CITYOBJECT_ID              INTEGER NOT NULL,
  CITYOBJECTGROUP_ID         INTEGER NOT NULL,
  ROLE                       VARCHAR(256)
); . . .
```

CITYMODEL.sql

```
CREATE TABLE CITYMODEL (
  ID                         SERIAL NOT NULL,
  GMLID                      VARCHAR(256),
  GMLID_CODESPACE            VARCHAR(1000),
  NAME                       VARCHAR(1000),
  NAME_CODESPACE             VARCHAR(4000),
  DESCRIPTION                VARCHAR(4000),
  ENVELOPE                   GEOMETRY(PolygonZ,:SRSNO),
  CREATION_DATE              DATE,
  TERMINATION_DATE           DATE,
  LAST_MODIFICATION_DATE     DATE,
  UPDATING_PERSON            VARCHAR(256),
  REASON_FOR_UPDATE          VARCHAR(4000),
  LINEAGE                    VARCHAR(256)
); . . .
```

CITYOBJECT_MEMBER.sql

```
CREATE TABLE CITYOBJECT_MEMBER (
  CITYMODEL_ID             INTEGER NOT NULL,
  CITYOBJECT_ID            INTEGER NOT NULL
); . . .
```

Oracle

CITYOBJECT.sql

```
CREATE TABLE CITYOBJECT (
  ID                          NUMBER NOT NULL,
  CLASS_ID                    NUMBER NOT NULL,
  GMLID                       VARCHAR2(256),
  GMLID_CODESPACE             VARCHAR2(1000),
  ENVELOPE                    MDSYS.SDO_GEOMETRY,
  CREATION_DATE               DATE NOT NULL,
  TERMINATION_DATE            DATE,
  LAST_MODIFICATION_DATE      DATE,
  UPDATING_PERSON             VARCHAR2(256),
  REASON_FOR_UPDATE           VARCHAR2(4000),
  LINEAGE                     VARCHAR2(256),
  XML_SOURCE                  CLOB
); . . .
```

EXTERNAL_REFERENCE.sql

```
CREATE TABLE EXTERNAL_REFERENCE (
  ID                          NUMBER NOT NULL,
  INFOSYS                     VARCHAR2(4000),
  NAME                        VARCHAR2(4000),
  URI                         VARCHAR2(4000),
  CITYOBJECT_ID               NUMBER NOT NULL
); . . .
```

GENERALIZATION.sql

```
CREATE TABLE GENERALIZATION (
  CITYMODEL_ID                NUMBER NOT NULL,
  GENERALIZES_TO_ID           NUMBER NOT NULL
); . . .
```

IMPLICIT_GEOMETRY.sql

```
CREATE TABLE IMPLICIT_GEOMETRY (
  ID                          NUMBER NOT NULL,
  MIME_TYPE                   VARCHAR2(256),
  REFERENCES_TO_LIBRARY       VARCHAR2(4000),
  LIBRARY_OBJECT              BLOB,
  RELATIVE_GEOMETRY_ID        NUMBER
); . . .
```

PostgreSQL / PostGIS

CITYOBJECT.sql

```
CREATE TABLE CITYOBJECT (
  ID                        SERIAL NOT NULL,
  CLASS_ID                  INTEGER NOT NULL,
  GMLID                     VARCHAR(256),
  GMLID_CODESPACE           VARCHAR(1000),
  ENVELOPE                  GEOMETRY(PolygonZ,:SRSNO),
  CREATION_DATE             DATE NOT NULL,
  TERMINATION_DATE          DATE,
  LAST_MODIFICATION_DATE    DATE,
  UPDATING_PERSON           VARCHAR(256),
  REASON_FOR_UPDATE         VARCHAR(4000),
  LINEAGE                   VARCHAR(256),
  XML_SOURCE                TEXT
); . . .
```

EXTERNAL_REFERENCE.sql

```
CREATE TABLE EXTERNAL_REFERENCE (
  ID                        SERIAL NOT NULL,
  INFOSYS                   VARCHAR(4000),
  NAME                      VARCHAR(4000),
  URI                       VARCHAR(4000),
  CITYOBJECT_ID             INTEGER NOT NULL
); . . .
```

GENERALIZATION.sql

```
CREATE TABLE GENERALIZATION (
  CITYMODEL_ID              INTEGER NOT NULL,
  GENERALIZES_TO_ID         INTEGER NOT NULL
); . . .
```

IMPLICIT_GEOMETRY.sql

```
CREATE TABLE IMPLICIT_GEOMETRY (
  ID                        SERIAL NOT NULL,
  MIME_TYPE                 VARCHAR(256),
  REFERENCES_TO_LIBRARY     VARCHAR(4000),
  LIBRARY_OBJECT            BYTEA,
  RELATIVE_GEOMETRY_ID      INTEGER
); . . .
```

Oracle

CITYOBJECT_GENERICATTRIB.sql

```
CREATE TABLE CITYOBJECT_GENERICATTRIB (
  ID                              NUMBER NOT NULL,
  ATTRNAME                        VARCHAR2(256) NOT NULL,
  DATATYPE                        NUMBER(1),
  STRVAL                          VARCHAR2(4000),
  INTVAL                          NUMBER,
  REALVAL                         NUMBER,
  URIVAL                          VARCHAR2(4000),
  DATEVAL                         DATE,
  GEOMVAL                         MDSYS.SDO_GEOMETRY,
  BLOBVAL                         BLOB,
  CITYOBJECT_ID                   NUMBER NOT NULL,
  SURFACE_GEOMETRY_ID             NUMBER
); . . .
```

GENERIC_CITYOBJECT.sql

```
CREATE TABLE GENERIC_CITYOBJECT (
  ID                              NUMBER NOT NULL,
  NAME                            VARCHAR2(1000),
  NAME_CODESPACE                  VARCHAR2(4000),
  DESCRIPTION                     VARCHAR2(4000),
  CLASS                           VARCHAR2(256),
  FUNCTION                        VARCHAR2(1000),
  USAGE                           VARCHAR2(1000),
  LOD0_TERRAIN_INTERSECTION       MDSYS.SDO_GEOMETRY, . . . -- auch für LOD 1,2,3,4
  LOD0_GEOMETRY_ID                NUMBER,             . . . -- auch für LOD 1,2,3,4
  LOD0_IMPLICIT_REP_ID            NUMBER,             . . . -- auch für LOD 1,2,3,4
  LOD0_IMPLICIT_REF_POINT         MDSYS.SDO_GEOMETRY, . . . -- auch für LOD 1,2,3,4
  LOD4_IMPLICIT_TRANSFORMATION    VARCHAR2(1000)      . . . -- auch für LOD 0,1,2,3
); . . .
```

SURFACE_GEOMETRY.sql

```
CREATE TABLE SURFACE_GEOMETRY (
  ID                              NUMBER NOT NULL,
  GMLID                           VARCHAR2(256),
  GMLID_CODESPACE                 VARCHAR2(1000),
  PARENT_ID                       NUMBER,
  ROOT_ID                         NUMBER,
  IS_SOLID                        NUMBER(1, 0),
  IS_COMPOSITE                    NUMBER(1, 0),
  IS_TRIANGULATED                 NUMBER(1, 0),
  IS_XLINK                        NUMBER(1, 0),
  IS_REVERSE                      NUMBER(1, 0),
  GEOMETRY                        MDSYS.SDO_GEOMETRY,
); . . .
```

PostgreSQL / PostGIS

CITYOBJECT_GENERICATTRIB.sql

```
CREATE TABLE CITYOBJECT_GENERICATTRIB (
  ID                          SERIAL NOT NULL,
  ATTRNAME                    VARCHAR(256) NOT NULL,
  DATATYPE                    NUMERIC(1),
  STRVAL                      VARCHAR(4000),
  INTVAL                      NUMERIC,
  REALVAL                     NUMERIC,
  URIVAL                      VARCHAR(4000),
  DATEVAL                     DATE,
  GEOMVAL                     GEOMETRY(GeometryZ,:SRSNO),
  BLOBVAL                     BYTEA,
  CITYOBJECT_ID               INTEGER NOT NULL,
  SURFACE_GEOMETRY_ID         INTEGER
); . . .
```

GENERIC_CITYOBJECT.sql

```
CREATE TABLE GENERIC_CITYOBJECT (
  ID                          SERIAL NOT NULL,
  NAME                        VARCHAR(1000),
  NAME_CODESPACE              VARCHAR(4000),
  DESCRIPTION                 VARCHAR(4000),
  CLASS                       VARCHAR(256),
  FUNCTION                    VARCHAR(1000),
  USAGE                       VARCHAR(1000),
  LOD0_TERRAIN_INTERSECTION   GEOMETRY(GeometryZ,:SRSNO), . . . -- siehe Links
  LOD0_GEOMETRY_ID            INTEGER,
  LOD0_IMPLICIT_REP_ID        INTEGER,
  LOD0_IMPLICIT_REF_POINT     GEOMETRY(PointZ,:SRSNO),
  LOD4_IMPLICIT_TRANSFORMATION VARCHAR(1000)
); . . .
```

SURFACE_GEOMETRY.sql

```
CREATE TABLE SURFACE_GEOMETRY (
  ID                          NUMBER NOT NULL,
  GMLID                       VARCHAR(256),
  GMLID_CODESPACE             VARCHAR(1000),
  PARENT_ID                   INTEGER,
  ROOT_ID                     INTEGER,
  IS_SOLID                    NUMERIC(1, 0),
  IS_COMPOSITE                NUMERIC(1, 0),
  IS_TRIANGULATED             NUMERIC(1, 0),
  IS_XLINK                    NUMERIC(1, 0),
  IS_REVERSE                  NUMERIC(1, 0),
  GEOMETRY                    GEOMETRY(PolygonZ,:SRSNO)
); . . .
```

Oracle

LAND_USE.sql

```
CREATE TABLE LAND_USE (
  ID                      NUMBER NOT NULL,
  NAME                    VARCHAR2(1000),
  NAME_CODESPACE          VARCHAR2(4000),
  DESCRIPTION             VARCHAR2(4000),
  CLASS                   VARCHAR2(256),
  FUNCTION                VARCHAR2(1000),
  USAGE                   VARCHAR2(1000),
  LOD0_MULTI_SURFACE_ID   NUMBER);          . . . -- auch für LOD 1,2,3,4
```

DATABASE_SRS.sql

```
CREATE TABLE DATABASE_SRS (
  SRID                    NUMBER NOT NULL,
  GML_SRS_NAME            VARCHAR2(1000)); . . .
```

OBJECTCLASS.sql

```
CREATE TABLE OBJECTCLASS (
  ID                      NUMBER NOT NULL,
  CLASSNAME               VARCHAR2(256),
  SUPERCLASS_ID           NUMBER); . . .
```

ORTHOPHOTO.sql

```
CREATE TABLE ORTHOPHOTO (
  ID                      NUMBER NOT NULL,
  LOD                     NUMBER (1) NOT NULL,
  NAME                    VARCHAR2(256),
  TYPE                    VARCHAR2(256),
  DATUM                   DATE,
  ORTHOPHOTOPROPERTY      MDSYS.SDO_GEORASTER); . . .
```

ORTHOPHOTO_IMP.sql

```
CREATE TABLE ORTHOPHOTO_IMP (
  ID                      NUMBER NOT NULL,
  ORTHOPHOTOPROPERTY      MDSYS.SDO_GEORASTER
  FILENAME                VARCHAR2(4000),
  FOOTPRINT               MDSYS.SDO_GEOMETRY); . . .
```

ORTHOPHOTO_RDT.sql / ORTHOPHOTO_RDT_IMP.sql

```
CREATE TABLE ORTHOPHOTO_RDT / ORTHOPHOTO_RDT_IMP OF SDO_RASTER (
  PRIMARY KEY (RASTERID, PYRAMIDLEVEL, BANDBLOCKNUMBER, ROWBLOCKNUMBER,
    COLUMNBLOCKNUMBER) LOB(RASTERBLOCK) STORE AS (NOCACHE NOLOGGING);
```

PostgreSQL / PostGIS

LAND_USE.sql

```
CREATE TABLE LAND_USE (
  ID                         SERIAL NOT NULL,
  NAME                       VARCHAR(1000),
  NAME_CODESPACE             VARCHAR(4000),
  DESCRIPTION                VARCHAR(4000),
  CLASS                      VARCHAR(256),
  FUNCTION                   VARCHAR(1000),
  USAGE                      VARCHAR(1000),
  LOD0_MULTI_SURFACE_ID      INTEGER,          . . . -- siehe links
);
```

DATABASE_SRS.sql

```
CREATE TABLE DATABASE_SRS (
  SRID                       INTEGER NOT NULL,
  GML_SRS_NAME               VARCHAR(1000)
); . . .
```

OBJECTCLASS.sql

```
CREATE TABLE OBJECTCLASS (
  ID                         SERIAL NOT NULL,
  CLASSNAME                  VARCHAR(256),
  SUPERCLASS_ID              INTEGER
); . . .
```

ORTHOPHOTO.sql

```
CREATE TABLE ORTHOPHOTO (
  ID                         SERIAL NOT NULL,
  LOD                        NUMERIC (1) NOT NULL,
  DATUM                      DATE,
  ORTHOPHOTOPROPERTY         RASTER
); . . .

-- Wenn raster2pgsql für den Import von Rasterdaten verwendet wird, sollte der
-- Operator -F verwendet werden, der eine Spalte mit dem Namen der Datei anlegt.
-- Daher wurde an dieser Stelle keine entsprechende Spalte angelegt. Die Spalte
-- Type erübrigt sich dadurch, dass die hinzugefügte von raster2pgsql Spalte auch
-- die Dateiendung enthält. Werte für die Spalten LOD und DATUM müssen nachträglich
-- in das von raster2pgsql generierte SQL-Skript eingefügt werden.

-- Wie in der Arbeit erwähnt, sind keine RDT- und IMP-Tabellen wie in der
-- Oracle-Version notwendig. Die Metadaten (u.a. auch Footprint) der Rasterdatei
-- werden in den Views raster_columns und raster_overviews gespeichert.
```

Oracle

BREAKLINE_RELIEF.sql

```
CREATE TABLE BREAKLINE_RELIEF (
  ID                        NUMBER NOT NULL,
  RIDGE_OR_VALLEY_LINES     MDSYS.SDO_GEOMETRY,
  BREAK_LINES               MDSYS.SDO_GEOMETRY
); . . .
```

MASSPOINT_RELIEF.sql

```
CREATE TABLE MASSPOINT_RELIEF (
  ID                        NUMBER NOT NULL,
  RELIEF_POINTS             MDSYS.SDO_GEOMETRY
); . . .
```

RASTER_RELIEF.sql

```
CREATE TABLE RASTER_RELIEF (
  ID                        NUMBER NOT NULL,
  -- LOD                    NUMBER (1) NOT NULL,
  RASTERPROPERTY            MDSYS.SDO_GEORASTER NOT NULL,
  -- RELIEF_ID              NUMBER NOT NULL,
  -- NAME                   VARCHAR2(256),
  -- TYPE                   VARCHAR2(256),
  -- EXTENT                 MDSYS.SDO_GEOMETRY
); . . .
```

RASTER_RELIEF_IMP.sql

```
CREATE TABLE RASTER_RELIEF_IMP (
  ID                        NUMBER NOT NULL,
  RASTERPROPERTY            MDSYS.SDO_GEORASTER,
  RELIEF_ID                 NUMBER,
  RASTER_RELIEF_ID          NUMBER,
  FILENAME                  VARCHAR2(4000),
  FOOTPRINT                 MDSYS.SDO_GEOMETRY
); . . .
```

RASTER_RELIEF_RDT.sql / RASTER_RELIEF_IMP_RDT.sql

```
CREATE TABLE RASTER_RELIEF_RDT / RASTER_RELIEF_IMP_RDT OF SDO_RASTER (
  PRIMARY KEY (RASTERID, PYRAMIDLEVEL, BANDBLOCKNUMBER, ROWBLOCKNUMBER,
    COLUMNBLOCKNUMBER) LOB(RASTERBLOCK) STORE AS (NOCACHE NOLOGGING);
```

PostgreSQL / PostGIS

BREAKLINE_RELIEF.sql

```
CREATE TABLE BREAKLINE_RELIEF (
  ID                       SERIAL NOT NULL,
  RIDGE_OR_VALLEY_LINES    GEOMETRY(MultiCurveZ,:SRSNO),
  BREAK_LINES              GEOMETRY(MultiCurveZ,:SRSNO)
); . . .
```

MASSPOINT_RELIEF.sql

```
CREATE TABLE MASSPOINT_RELIEF (
  ID                       SERIAL NOT NULL,
  RELIEF_POINTS            GEOMETRY(MultiPointZ,:SRSNO),
); . . .
```

RASTER_RELIEF.sql

```
CREATE TABLE RASTER_RELIEF (
  ID                       SERIAL NOT NULL,
  LOD                      NUMERIC (1) NOT NULL,
  RASTERPROPERTY           RASTER NOT NULL,
  RELIEF_ID                INTEGER NOT NULL
); . . .

-- Wie bei ORTHOPHOTO wird nur noch eine RASTER_RELIEF Tabelle benötigt. Die Spalten
-- NAME, TYPE, FILENAME, FOOTPRINT können wie oben beschrieben entfallen. Für die
-- Spalte Relief_ID wurde (in CONSTRAINTS.sql) ein Foreign Key zur Tabelle RELIEF
-- definiert.
```

Oracle

RELIEF.sql

```
CREATE TABLE RELIEF (
  ID                        NUMBER NOT NULL,
  NAME                      VARCHAR2(256),
  TYPE                      VARCHAR2(256),
  LOD                       NUMBER (1) NOT NULL
); . . .
```

RELIEF_COMPONENT.sql

```
CREATE TABLE RELIEF_COMPONENT (
  ID                        NUMBER NOT NULL,
  NAME                      VARCHAR2(1000),
  NAME_CODESPACE            VARCHAR2(4000),
  DESCRIPTION               VARCHAR2(4000),
  LOD                       NUMBER(1),
  EXTENT                    MDSYS.SDO_GEOMETRY
); . . .
```

RELIEF_FEAT_TO_REL_COMP.sql

```
CREATE TABLE RELIEF_FEAT_TO_REL_COMP (
  RELIEF_COMPONENT_ID       NUMBER NOT NULL,
  RELIEF_FEATURE_ID         NUMBER NOT NULL
); . . .
```

RELIEF_FEATURE.sql

```
CREATE TABLE RELIEF_FEATURE (
  ID                        NUMBER NOT NULL,
  NAME                      VARCHAR2(1000),
  NAME_CODESPACE            VARCHAR2(4000),
  DESCRIPTION               VARCHAR2(4000),
  LOD                       NUMBER(1)
); . . .
```

TIN_RELIEF.sql

```
CREATE TABLE TIN_RELIEF (
  ID                        NUMBER NOT NULL,
  MAX_LENGTH                BINARY_DOUBLE,
  STOP_LINES                MDSYS.SDO_GEOMETRY,
  BREAK_LINES               MDSYS.SDO_GEOMETRY,
  CONTROL_POINTS            MDSYS.SDO_GEOMETRY,
  SURFACE_GEOMETRY_ID       NUMBER
); . . .
```

PostgreSQL / PostGIS

RELIEF.sql

```
CREATE TABLE RELIEF (
  ID                      SERIAL NOT NULL,
  NAME                    VARCHAR(256),
  TYPE                    VARCHAR(256),
  LOD                     NUMERIC (1) NOT NULL
); . . .
```

RELIEF_COMPONENT.sql

```
CREATE TABLE RELIEF_COMPONENT (
  ID                      NUMBER NOT NULL,
  NAME                    VARCHAR(1000),
  NAME_CODESPACE          VARCHAR(4000),
  DESCRIPTION             VARCHAR(4000),
  LOD                     NUMERIC(1),
  EXTENT                  GEOMETRY(Polygon,:SRSNO)
); . . .
```

RELIEF_FEAT_TO_REL_COMP.sql

```
CREATE TABLE RELIEF_FEAT_TO_REL_COMP (
  RELIEF_COMPONENT_ID     INTEGER NOT NULL,
  RELIEF_FEATURE_ID       INTEGER NOT NULL
); . . .
```

RELIEF_FEATURE.sql

```
CREATE TABLE RELIEF_FEATURE (
  ID                      SERIAL NOT NULL,
  NAME                    VARCHAR(1000),
  NAME_CODESPACE          VARCHAR(4000),
  DESCRIPTION             VARCHAR(4000),
  LOD                     NUMERIC(1)
); . . .
```

TIN_RELIEF.sql

```
CREATE TABLE TIN_RELIEF (
  ID                      SERIAL NOT NULL,
  MAX_LENGTH              DOUBLE PRECISION,
  STOP_LINES              GEOMETRY(MultiCurveZ,:SRSNO),
  BREAK_LINES             GEOMETRY(MultiCurveZ,:SRSNO),
  CONTROL_POINTS          GEOMETRY(MultiPointZ,:SRSNO),
  SURFACE_GEOMETRY_ID     INTEGER
); . . .
```

Oracle

TRAFFIC_AREA.sql

```
CREATE TABLE TRAFFIC_AREA (
  ID                          NUMBER NOT NULL,
  IS_AUXILIARY                NUMBER(1),
  NAME                        VARCHAR2(1000),
  NAME_CODESPACE              VARCHAR2(4000),
  DESCRIPTION                 VARCHAR2(4000),
  FUNCTION                    VARCHAR2(1000),
  USAGE                       VARCHAR2(1000),
  SURFACE_MATERIAL            VARCHAR2(256),
  LOD2_MULTI_SURFACE_ID       NUMBER,                   . . . -- auch für LOD 3,4
  TRANSPORTATION_COMPLEX_ID   NUMBER NOT NULL,
); . . .
```

TRANSPORTATION_COMPLEX.sql

```
CREATE TABLE TRANSPORTATION_COMPLEX (
  ID                          NUMBER NOT NULL,
  NAME                        VARCHAR2(1000),
  NAME_CODESPACE              VARCHAR2(4000),
  DESCRIPTION                 VARCHAR2(4000),
  FUNCTION                    VARCHAR2(1000),
  USAGE                       VARCHAR2(1000),
  TYPE                        VARCHAR2(256),
  LOD0_NETWORK                MDSYS.SDO_GEOMETRY,
  LOD1_MULTI_SURFACE_ID       NUMBER,                   . . . -- auch für LOD 2,3,4
); . . .
```

PLANT_COVER.sql

```
CREATE TABLE PLANT_COVER (
  ID                          NUMBER NOT NULL,
  NAME                        VARCHAR2(1000),
  NAME_CODESPACE              VARCHAR2(4000),
  DESCRIPTION                 VARCHAR2(4000),
  CLASS                       VARCHAR2(256),
  FUNCTION                    VARCHAR2(1000),
  AVERAGE_HEIGHT              BINARY_DOUBLE,
  LOD1_GEOMETRY_ID            NUMBER,                   . . . -- auch für LOD 2,3,4
); . . .
```

PostgreSQL / PostGIS

TRAFFIC_AREA.sql

```
CREATE TABLE TRAFFIC_AREA (
  ID                          SERIAL NOT NULL,
  IS_AUXILIARY                NUMERIC(1),
  NAME                        VARCHAR(1000),
  NAME_CODESPACE              VARCHAR(4000),
  DESCRIPTION                 VARCHAR(4000),
  FUNCTION                    VARCHAR(1000),
  USAGE                       VARCHAR(1000),
  SURFACE_MATERIAL            VARCHAR(256),
  LOD2_MULTI_SURFACE_ID       INTEGER,                    . . . -- siehe links
  TRANSPORTATION_COMPLEX_ID   INTEGER NOT NULL,
); . . .
```

TRANSPORTATION_COMPLEX.sql

```
CREATE TABLE TRANSPORTATION_COMPLEX (
  ID                          SERIAL NOT NULL,
  NAME                        VARCHAR(1000),
  NAME_CODESPACE              VARCHAR(4000),
  DESCRIPTION                 VARCHAR(4000),
  FUNCTION                    VARCHAR(1000),
  USAGE                       VARCHAR(1000),
  TYPE                        VARCHAR(256),
  LOD0_NETWORK                GEOMETRY(GeometryZ,:SRSNO),
  LOD1_MULTI_SURFACE_ID       INTEGER,                    . . . -- siehe links
); . . .
```

PLANT_COVER.sql

```
CREATE TABLE PLANT_COVER (
  ID                          SERIAL NOT NULL,
  NAME                        VARCHAR(1000),
  NAME_CODESPACE              VARCHAR(4000),
  DESCRIPTION                 VARCHAR(4000),
  CLASS                       VARCHAR(256),
  FUNCTION                    VARCHAR(1000),
  AVERAGE_HEIGHT              DOUBLE PRECISION,
  LOD1_GEOMETRY_ID            INTEGER,                    . . . -- siehe links
); . . .
```

Oracle

SOLITARY_VEGETAT_OBJECT.sql

```
CREATE TABLE SOLITARY_VEGETAT_OBJECT (
  ID                              NUMBER NOT NULL,
  NAME                            VARCHAR2(1000),
  NAME_CODESPACE                  VARCHAR2(4000),
  DESCRIPTION                     VARCHAR2(4000),
  CLASS                           VARCHAR2(256),
  SPECIES                         VARCHAR2(1000),
  FUNCTION                        VARCHAR2(1000),
  HEIGHT                          BINARY_DOUBLE,
  TRUNC_DIAMETER                  BINARY_DOUBLE,
  CROWN_DIAMETER                  BINARY_DOUBLE,
  LOD1_GEOMETRY_ID                NUMBER,              . . . -- auch für LOD 2,3,4
  LOD1_IMPLICIT_REP_ID            NUMBER,              . . . -- auch für LOD 2,3,4
  LOD1_IMPLICIT_REF_POINT         MDSYS.SDO_GEOMETRY,  . . . -- auch für LOD 2,3,4
  LOD4_IMPLICIT_TRANSFORMATION    VARCHAR2(1000));     . . . -- auch für LOD 1,2,3
); . . .
```

WATERBOD_TO_WATERBND_SRF.sql

```
CREATE TABLE WATERBOD_TO_WATERBND_SRF (
  WATERBOUNDARY_SURFACE_ID      NUMBER NOT NULL,
  WATERBODY_ID                  NUMBER NOT NULL); . . .
```

WATERBODY.sql

```
CREATE TABLE WATERBODY (
  ID                              NUMBER NOT NULL,
  NAME                            VARCHAR2(1000),
  NAME_CODESPACE                  VARCHAR2(4000),
  DESCRIPTION                     VARCHAR2(4000),
  CLASS                           VARCHAR2(256),
  FUNCTION                        VARCHAR2(1000),
  USAGE                           VARCHAR2(1000),
  LOD0_MULTI_CURVE                MDSYS.SDO_GEOMETRY,   . . . -- auch für LOD 1
  LOD1_SOLID_ID                   NUMBER,               . . . -- auch für LOD 2,3,4
  LOD0_MULTI_SURFACE_ID           NUMBER);              . . . -- auch für LOD 1
```

WATERBOUNDARY_SURFACE.sql

```
CREATE TABLE WATERBOUNDARY_SURFACE (
  ID                              NUMBER NOT NULL,
  NAME                            VARCHAR2(1000),
  NAME_CODESPACE                  VARCHAR2(4000),
  DESCRIPTION                     VARCHAR2(4000),
  TYPE                            VARCHAR2(256),
  WATER_LEVEL                     VARCHAR2(256),
  LOD2_SURFACE_ID                 NUMBER);              . . . -- auch für LOD 3,4
```

PostgreSQL / PostGIS

SOLITARY_VEGETAT_OBJECT.sql

```
CREATE TABLE SOLITARY_VEGETAT_OBJECT (
  ID                            SERIAL NOT NULL,
  NAME                          VARCHAR(1000),
  NAME_CODESPACE                VARCHAR(4000),
  DESCRIPTION                   VARCHAR(4000),
  CLASS                         VARCHAR(256),
  SPECIES                       VARCHAR(1000),
  FUNCTION                      VARCHAR(1000),
  HEIGHT                        DOUBLE PRECISION,
  TRUNC_DIAMETER                DOUBLE PRECISION,
  CROWN_DIAMETER                DOUBLE PRECISION,
  LOD1_GEOMETRY_ID              INTEGER,                   . . . -- siehe links
  LOD1_IMPLICIT_REP_ID          INTEGER,                   . . .
  LOD1_IMPLICIT_REF_POINT       GEOMETRY(PointZ,:SRSNO),   . . .
  LOD4_IMPLICIT_TRANSFORMATION  VARCHAR(1000));            . . .
); . . .
```

WATERBOD_TO_WATERBND_SRF.sql

```
CREATE TABLE WATERBOD_TO_WATERBND_SRF (
  WATERBOUNDARY_SURFACE_ID     INTEGER NOT NULL,
  WATERBODY_ID                 INTEGER NOT NULL); . . .
```

WATERBODY.sql

```
CREATE TABLE WATERBODY (
  ID                            SERIAL NOT NULL,
  NAME                          VARCHAR(1000),
  NAME_CODESPACE                VARCHAR(4000),
  DESCRIPTION                   VARCHAR(4000),
  CLASS                         VARCHAR(256),
  FUNCTION                      VARCHAR(1000),
  USAGE                         VARCHAR(1000),
  LOD0_MULTI_CURVE              GEOMETRY(MultiCurveZ,:SRSNO),   . . . -- siehe links
  LOD1_SOLID_ID                 INTEGER,                        . . .
  LOD0_MULTI_SURFACE_ID         INTEGER);                       . . .
```

WATERBOUNDARY_SURFACE.sql

```
CREATE TABLE WATERBOUNDARY_SURFACE (
  ID                            SERIAL NOT NULL,
  NAME                          VARCHAR(1000),
  NAME_CODESPACE                VARCHAR(4000),
  DESCRIPTION                   VARCHAR(4000),
  TYPE                          VARCHAR(256),
  WATER_LEVEL                   VARCHAR(256),
  LOD2_SURFACE_ID               INTEGER);                  . . . -- siehe links
```

A.3. Hilfsskripte

Tab. A4: Der Inhalt vom UTIL-Ordner im Überblick

3dcitydb/postgis/UTIL			Oracle	PgSQL	Anmerkungen
		CREATE_DB			
		CHECK_SRID.sql	X	✓	Alternative zu HINT_ON_MISSING_SRS
		DO_NOTHING.sql	✓	X	wird ausgeführt, wenn keine Versionierung vorgenommen werden sollte
		DUMMY_IMPORT.sql	✓	X	diente dem Raster-Import in Oracle
		HINT_ON_MISSING_SRS.sql	✓	X	
		IMPORT_PROCEDURES.sql	✓	X	diente dem Raster-Import in Oracle
		OBJECTCLASS_INSTANCES.sql	✓	✓	
		RO_USER			
		CREATE_RO_USER.sql	✓	✓	
		SRS			
		SOLDNER_BERLIN_SRS.sql	✓	X	Soldner-Referenz-System wird in Oracle nicht per default angeboten
		VACUUM			
		VACUUM_SPATIAL_COLUMNS.sql	X	✓	PostGIS-spezifisch

CHECK_SRID.sql

```
CREATE OR REPLACE FUNCTION check_srid(srid INTEGER DEFAULT 0) RETURNS VARCHAR AS
$$
DECLARE
  dbsrid INTEGER;
  validation VARCHAR := 'SRID ok';
BEGIN
  EXECUTE 'SELECT srid FROM spatial_ref_sys WHERE srid = $1' INTO dbsrid USING
srid;

  IF dbsrid <> 0 THEN
    BEGIN
      PERFORM ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(1 1 1)'),dbsrid);

        RETURN validation;

      EXCEPTION
        WHEN others THEN
          RAISE EXCEPTION 'The chosen SRID % was not appropriate for PostGIS
             functions.', srid;
    END;
  ELSE
    RAISE EXCEPTION 'Table spatial_ref_sys does not contain the SRID %. Insert
      commands for missing SRIDs can be found at spatialreference.org', srid;
  END IF;
END; $$ LANGUAGE plpgsql;
```

OBJECTCLASS_INSTANCES.sql (kein Unterschied zur Oracle-Version)

```
DELETE FROM OBJECTCLASS;

INSERT INTO OBJECTCLASS (ID,CLASSNAME,SUPERCLASS_ID) VALUES (0,'Undefined',NULL);
INSERT INTO OBJECTCLASS (ID,CLASSNAME,SUPERCLASS_ID) VALUES (1,'Object',NULL);
. . .
```

CREATE_RO_USER.sql (Oracle-Version)

```
SET SERVEROUTPUT ON;
-- SET FEEDBACK ON

prompt
prompt
accept RO_USERNAME CHAR PROMPT 'Please enter a username for the read-only user: '
accept SCHEMA_OWNER CHAR PROMPT 'Please enter the owner of the schema to which this
user will have read-only access: '
prompt
prompt

DECLARE
 v_schemaOwnerName ALL_USERS.USERNAME%TYPE := null;
 v_readOnlyName ALL_USERS.USERNAME%TYPE := null;
 v_role DBA_ROLES.ROLE%TYPE := null;

 RO_USER_ALREADY_EXISTS EXCEPTION;

BEGIN

  IF ('&RO_USERNAME' IS NULL) THEN
    dbms_output.put_line('Invalid username!');
  END IF;

  IF ('&SCHEMA_OWNER' IS NULL) THEN
    dbms_output.put_line('Invalid schema owner!');
  END IF;

  BEGIN
    SELECT USERNAME INTO v_schemaOwnerName FROM ALL_USERS WHERE USERNAME =
      UPPER('&SCHEMA_OWNER');

  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      dbms_output.put_line('Schema owner ' || '&SCHEMA_OWNER' || ' does not
                            exist!');
      RAISE;
  END;

  BEGIN
    SELECT USERNAME INTO v_readOnlyName FROM ALL_USERS WHERE USERNAME =
      UPPER('&RO_USERNAME');
```

```
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      NULL; -- do nothing, read-only user must not exist
    WHEN OTHERS THEN
      RAISE;
  END;

  IF (v_readOnlyName IS NOT NULL) THEN
    RAISE RO_USER_ALREADY_EXISTS;
  END IF;

  v_readOnlyName := '&RO_USERNAME';
  EXECUTE IMMEDIATE 'create user ' || v_readOnlyName || ' identified by berlin3d
                       PASSWORD EXPIRE';

  BEGIN
    SELECT ROLE INTO v_role FROM DBA_ROLES WHERE ROLE = UPPER('&SCHEMA_OWNER') ||
      '_READ_ONLY';
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      v_role := UPPER('&SCHEMA_OWNER') || '_READ_ONLY';
      EXECUTE IMMEDIATE 'create role ' || v_role;

      EXECUTE IMMEDIATE 'grant select on ' || UPPER('&SCHEMA_OWNER') || '.ADDRESS
                           to ' || v_role;
      . . .

    WHEN OTHERS THEN
      RAISE;
  END;

  EXECUTE IMMEDIATE 'grant ' || v_role || ' to ' || v_readOnlyName;
  EXECUTE IMMEDIATE 'grant CONNECT to ' || v_readOnlyName;
  EXECUTE IMMEDIATE 'grant RESOURCE to ' || v_readOnlyName;

-- synonyms for tables
  EXECUTE IMMEDIATE 'create or replace synonym ' || v_readOnlyName ||
    '.MATCH_RESULT_RELEVANT for ' || v_schemaOwnerName || '.MATCH_RESULT_RELEVANT';
  . . .

-- synonyms for PL/SQL packages
  EXECUTE IMMEDIATE 'create or replace synonym ' || v_readOnlyName ||
                      '.GEODB_DELETE for ' || v_schemaOwnerName || '.GEODB_DELETE';
  . . .

-- synonyms for user defined object types
  EXECUTE IMMEDIATE 'create or replace synonym ' || v_readOnlyName || '.INDEX_OBJ
                      for ' || v_schemaOwnerName || '.INDEX_OBJ';
  . . .

  COMMIT;
  dbms_output.put_line(' ');
  dbms_output.put_line('create_ro_user.sql finished successfully');
```

```
EXCEPTION
  WHEN RO_USER_ALREADY_EXISTS THEN
    dbms_output.put_line(' ');
    dbms_output.put_line('User ' || '&RO_USERNAME' || ' already exists!');
    dbms_output.put_line('create_ro_user.sql finished with errors');
  WHEN OTHERS THEN
    dbms_output.put_line(' ');
    dbms_output.put_line('create_ro_user.sql finished with errors');
END;
/
```

CREATE_RO_USER.sql (PostgreSQL-Version)

```
\prompt 'Please enter a username for the read-only user: ' RO_USERNAME
\prompt 'Please enter a password for the read-only user: ' RO_PASSWORD

CREATE ROLE :RO_USERNAME WITH NOINHERIT LOGIN PASSWORD :'RO_PASSWORD';

GRANT USAGE ON SCHEMA public TO :RO_USERNAME;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO :RO_USERNAME;
GRANT USAGE ON SCHEMA geodb_pkg TO :RO_USERNAME;
GRANT SELECT ON ALL TABLES IN SCHEMA geodb_pkg TO :RO_USERNAME;
```

VACUUM_SPATIAL_COLUMNS.sql

```
VACUUM ANALYSE cityobject (envelope);
VACUUM ANALYSE surface_geometry (geometry);
. . .
```

A.4. PL/pgSQL-Skripte

Tab. A5: Übersicht von PL/pgSQL-Skripten

3dcitydb/postgis/PL_pgSQL/GEODB_PKG	Oracle	PgSQL	Prefix für Funktionen
GEODB_PKG/DELETE			
DELETE.sql	✓	✓	del_
DELETE_BY_LINEAGE.sql	✓	✓	del_by_lin_
GEODB_PKG/INDEX			
IDX.sql	✓	✓	idx_
GEODB_PKG/MATCHING			
MATCH.sql	✓	✓	match_
MERGE.sql	✓	✓	merge_
GEODB_PKG/STATISTICS			
STAT.sql	✓	✓	stat_
GEODB_PKG/UTIL			
UTILsql	✓	✓	util_
MOSAIC			
MOSAIC.sql	✓	x	diente zur Bearbeitung von Rasterdaten in der Oracle-Version

Oracle

DELETE.sql

```
CREATE OR REPLACE PACKAGE geodb_delete
AS
  procedure delete_surface_geometry(pid number, clean_apps int := 0);
   . . .
END geodb_delete;
/

CREATE OR REPLACE PACKAGE BODY geodb_delete
AS
  -- private procedures
  procedure intern_delete_surface_geometry(pid number);
  . . .

  function is_not_referenced(table_name varchar2, check_column varchar2,
             check_id number, not_column varchar2, not_id number) return boolean;

  type ref_cursor is ref cursor;

  /* internal helpers */
  function is_not_referenced(
    table_name varchar2,
    check_column varchar2,
    check_id number,
    not_column varchar2,
    not_id number)
  return boolean
    is
    ref_cur ref_cursor;
    dummy number;
    is_not_referenced boolean;
  begin
    open ref_cur for 'select 1 from ' || table_name || ' where ' || check_column ||
      '=:1 and not ' || not_column || '=:2' using check_id, not_id;
    loop
      fetch ref_cur into dummy;

      is_not_referenced := ref_cur%notfound;



      exit;
    end loop;
    close ref_cur;

    return is_not_referenced;
  end;
```

PostgreSQL / PostGIS

DELETE.sql

```
-- In Kapitel 6.2.1. wurde das Fehlen von Packages für Stored Procedures in
-- PostgreSQL festgestellt und erklärt mit welchen Methoden sie substituiert wurden.
-- Es gibt demnach keine Aufteilung in Spezifikation und Quellcode-Körper, wie
-- links für die Oracle-Version zu sehen ist. Dies gilt für alle PL/pgSQL-Dateien.
```

```
-- In der PostGIS-Version werden alle Funktion durch die Angabe des Schemas im
-- Funktionsnamen in selbiges gespeichert.
```

```
/* internal helpers */
CREATE OR REPLACE FUNCTION geodb_pkg.del_is_not_referenced(
  table_name VARCHAR,
  check_column VARCHAR,
  check_id NUMERIC,
  not_column VARCHAR,
  not_id NUMERIC)
RETURNS BOOLEAN AS $$
DECLARE
  ref_cur refcursor;
  dummy NUMERIC;
  is_not_referenced BOOLEAN;
BEGIN
  OPEN ref_cur FOR EXECUTE 'SELECT 1 from ' || table_name || ' WHERE ' ||
    check_column || '=$1 and not ' || not_column || '=$2' USING check_id, not_id;
  LOOP
    FETCH ref_cur INTO dummy;
    IF NOT FOUND THEN
      is_not_referenced := true;
    ELSE
      is_not_referenced := false;
    END IF;
    EXIT;
  END LOOP;
  CLOSE ref_cur;

  RETURN is_not_referenced;
END; $$ LANGUAGE plpgsql;
```

Oracle

DELETE.sql (Fortführung)

```
/* internal: delete from SURFACE_GEOMETRY */

procedure intern_delete_surface_geometry(pid number)
is
begin
  execute immediate 'delete from textureparam where surface_geometry_id in
    (select id from (select id from surface_geometry start with id=:1
       connect by prior id=parent_id order by level desc))' using pid;
  -- start with ... connect by prior ist die Oracle-spezifische Lösung für eine
  -- rekursive Abfrage an die Datenbank. In diesem Fall soll verhindert werden,
  -- dass eine Elterngeometrie (parent_id) vor seinen Kindelementen (root_id)
  -- gelöscht wird.

  execute immediate 'delete from surface_geometry where id in (select id from
    (select id from surface_geometry start with id=:1 connect by prior
       id=parent_id order by level desc))' using pid;

  -- Die Lösung für PostgreSQL entspricht dem internationalen SQL Standard.


exception
  when others then
    dbms_output.put_line('intern_delete_surface_geometry (id: ' || pid || '): '
      || SQLERRM);
end;


/* internal: delete from IMPLICIT_GEOMETRY */
procedure intern_delete_implicit_geom(pid number)

is
  implicit_geometry_rec implicit_geometry%rowtype;
begin
  execute immediate 'select * from implicit_geometry where id=:1'
    into implicit_geometry_rec using pid;
  execute immediate 'delete from implicit_geometry where id=:1' using pid;

  post_delete_implicit_geom(implicit_geometry_rec);
  -- PostgreSQL erlaubt keine Cursor-Variablen in der Funktionsspezifikation.


exception
  when others then
    dbms_output.put_line('intern_delete_implicit_geom (id: ' || pid || '): ' ||
      SQLERRM);
end;
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
/* internal: DELETE FROM SURFACE_GEOMETRY */
-- in PL/pgSQL gibt es keine Prozeduren, RETURN SETOF void ist aber gleichbedeutend
CREATE OR REPLACE FUNCTION geodb_pkg.del_intern_delete_surface_geometry(pid NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  EXECUTE 'DELETE FROM textureparam WHERE surface_geometry_id IN
             (WITH RECURSIVE recursive_query(id, parent_id, level) AS
                (SELECT id, parent_id, 1 AS level FROM surface_geometry WHERE id=$1
              UNION ALL
                SELECT sg.id, sg.parent_id, rq.level + 1 AS level FROM
                surface_geometry sg, recursive_query rq WHERE sg.parent_id = rq.id)
              SELECT id FROM recursive_query ORDER BY level DESC)' USING pid;

  EXECUTE 'DELETE FROM surface_geometry WHERE id IN
             (WITH RECURSIVE recursive_query(id, parent_id, level) AS
                (SELECT id, parent_id, 1 AS level FROM surface_geometry WHERE id=$1
              UNION ALL
                SELECT sg.id, sg.parent_id, rq.level + 1 AS level FROM
                surface_geometry sg, recursive_query rq WHERE sg.parent_id = rq.id)
              SELECT id FROM recursive_query ORDER BY level DESC)' USING pid;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'intern_delete_surface_geometry (id: %): %', pid, SQLERRM;
  -- Das %-Zeichen kann in Meldungen als Platzhalter für Variablen verwendet werden.
END; $$ LANGUAGE plpgsql;


/* internal: DELETE FROM IMPLICIT_GEOMETRY */
CREATE OR REPLACE FUNCTION geodb_pkg.del_intern_delete_implicit_geom(pid NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  implicit_geometry_rec implicit_geometry%ROWTYPE;
BEGIN
  EXECUTE 'SELECT * FROM implicit_geometry WHERE id=$1'
    INTO implicit_geometry_rec USING pid;
  EXECUTE 'DELETE FROM implicit_geometry WHERE id=$1' USING pid;

  PERFORM geodb_pkg.del_post_delete_implicit_geom(
            implicit_geometry_rec.relative_geometry_id);
  -- Wenn Funktionsaufrufe kein Wert zurück erhalten, wird in PL/pgSQL PERFORM statt
  -- dem sonst für PL/pgSQL üblichen SELECT verwendet.
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'intern_delete_implicit_geom (id: %): %', pid, SQLERRM;

END; $$ LANGUAGE plpgsql;
```

Oracle

DELETE.sql (Fortführung)

```
procedure post_delete_implicit_geom(implicit_geometry_rec implicit_geometry%rowtype)

is
begin
  if implicit_geometry_rec.relative_geometry_id is not null then
    intern_delete_surface_geometry(implicit_geometry_rec.relative_geometry_id);
  end if;

exception
  when others then
    dbms_output.put_line('post_delete_implicit_geom (id: ' || implicit_geometry_rec.id
      || '): ' || SQLERRM);
end;

/*
  internal: delete from CITY_OBJECT
*/
procedure pre_delete_cityobject(pid number)

is
  cursor appearance_cur is
    select * from appearance where cityobject_id=pid;
begin
  execute immediate 'delete from cityobject_member where cityobject_id=:1' using pid;
  execute immediate 'delete from group_to_cityobject where cityobject_id=:1' using pid;
  execute immediate 'delete from generalization where generalizes_to_id=:1' using pid;
  execute immediate 'delete from generalization where cityobject_id=:1' using pid;
  execute immediate 'delete from external_reference where cityobject_id=:1' using pid;
  execute immediate 'delete from cityobject_genericattrib where cityobject_id=:1'
                    using pid;
  execute immediate 'update cityobjectgroup set parent_cityobject_id=null where
                    parent_cityobject_id=:1' using pid;

  for rec in appearance_cur loop
    delete_appearance(rec);
  end loop;

exception
  when others then
    dbms_output.put_line('pre_delete_cityobject (id: ' || pid || '): ' || SQLERRM);
end;


procedure intern_delete_cityobject(pid number)
is
begin
  pre_delete_cityobject(pid);
  execute immediate 'delete from cityobject where id=:1' using pid;

exception
  when others then
    dbms_output.put_line('intern_delete_cityobject (id: ' || pid || '): ' || SQLERRM);
end;
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION
geodb_pkg.del_post_delete_implicit_geom(relative_geometry_id NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  IF relative_geometry_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(relative_geometry_id);
  END IF;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'post_delete_implicit_geom (id: %): %', relative_geometry_id,
                      SQLERRM;
END; $$ LANGUAGE plpgsql;

/*
internal: DELETE FROM CITY_OBJECT
*/
CREATE OR REPLACE FUNCTION geodb_pkg.del_pre_delete_cityobject(pid NUMERIC) RETURNS
SETOF void AS $$
DECLARE
  appearance_cur CURSOR FOR
    SELECT * FROM appearance WHERE cityobject_id=pid;
BEGIN
  EXECUTE 'DELETE FROM cityobject_member WHERE cityobject_id=$1' USING pid;
  EXECUTE 'DELETE FROM group_to_cityobject WHERE cityobject_id=$1' USING pid;
  EXECUTE 'DELETE FROM generalization WHERE generalizes_to_id=$1' USING pid;
  EXECUTE 'DELETE FROM generalization WHERE cityobject_id=$1' USING pid;
  EXECUTE 'DELETE FROM external_reference WHERE cityobject_id=$1' USING pid;
  EXECUTE 'DELETE FROM cityobject_genericattrib WHERE cityobject_id=$1' USING pid;

  EXECUTE 'UPDATE cityobjectgroup SET parent_cityobject_id=null WHERE
             parent_cityobject_id=$1' USING pid;

  FOR rec IN appearance_cur LOOP
    PERFORM geodb_pkg.del_delete_row_appearance(rec.id);
  END LOOP;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'pre_delete_cityobject (id: %): %', pid, SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_intern_delete_cityobject(pid NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  PERFORM geodb_pkg.del_pre_delete_cityobject(pid);
  EXECUTE 'DELETE FROM cityobject WHERE id=$1' USING pid;

EXCEPTION
  WHEN OTHERS THEN
    RAISE NOTICE 'intern_delete_cityobject (id: %): %', pid, SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

DELETE.sql (Fortführung)

```
/* internal: delete from CITYMODEL */
procedure pre_delete_citymodel(citymodel_rec citymodel%rowtype)

is
  cursor appearance_cur is
    select * from appearance where cityobject_id=citymodel_rec.id;
begin
  -- TODO delete contained cityobjects!
  execute immediate 'delete from cityobject_member where citymodel_id=:1'
                    using citymodel_rec.id;

  for rec in appearance_cur loop
    delete_appearance(rec);
  end loop;
exception
  when others then
    dbms_output.put_line('pre_delete_citymodel (id: ' || citymodel_rec.id ||
      '): ' || SQLERRM);
end;



procedure delete_citymodel(citymodel_rec citymodel%rowtype)
is
begin
  pre_delete_citymodel(citymodel_rec);
  execute immediate 'delete from citymodel where id=:1' using citymodel_rec.id;
exception
  when others then
    dbms_output.put_line('delete_citymodel (id: ' || citymodel_rec.id || '): ' ||
      SQLERRM);
end;


/* internal: delete from APPEARANCE */
procedure pre_delete_appearance(appearance_rec appearance%rowtype)

is
  cursor surface_data_cur is
    select s.* from surface_data s, appear_to_surface_data ats
      where s.id=ats.surface_data_id and ats.appearance_id=appearance_rec.id;
begin
  -- delete surface data not being referenced by appearances any more
  for rec in surface_data_cur loop
    if is_not_referenced('appear_to_surface_data', 'surface_data_id',
      rec.id, 'appearance_id', appearance_rec.id) then
         delete_surface_data(rec);
    end if;
  end loop;

  execute immediate 'delete from appear_to_surface_data where appearance_id=:1'
                    using appearance_rec.id;
exception
  when others then
    dbms_output.put_line('pre_delete_appearance (id: ' || appearance_rec.id ||
      '): ' || SQLERRM);
end;
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
/* internal: DELETE FROM CITYMODEL */
CREATE OR REPLACE FUNCTION geodb_pkg.del_pre_delete_citymodel(citymodel_row_id
NUMERIC) RETURNS SETOF void AS $$
DECLARE
  appearance_cur CURSOR FOR
    SELECT * FROM appearance WHERE cityobject_id=citymodel_row_id;
BEGIN
-- TODO delete contained cityobjects!
  EXECUTE 'DELETE FROM cityobject_member WHERE citymodel_id=$1'
             USING citymodel_row_id;
  FOR rec IN appearance_cur LOOP
    PERFORM geodb_pkg.del_delete_row_appearance(rec.id);
    -- Da nur die ID übergeben wird, würde es zu Namensdoppelung bei den Methoden
    -- kommen. Deshalb wurde der Zusatz „_row_“ hinzugefügt in Anlehnung daran, dass
    -- vormals eine Cursor-Variable mit den Werten einer Zeile übergeben wurde.
  END LOOP;
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'pre_delete_citymodel (id: %): %', citymodel_row_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_delete_row_citymodel(citymodel_id NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  PERFORM geodb_pkg.del_pre_delete_citymodel(citymodel_id);
  EXECUTE 'DELETE FROM citymodel WHERE id=$1' USING citymodel_id;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_row_citymodel (id: %): %', citymodel_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


/* internal: DELETE FROM APPEARANCE */
CREATE OR REPLACE FUNCTION geodb_pkg.del_pre_delete_appearance(appearance_row_id
NUMERIC) RETURNS SETOF void AS $$
DECLARE
  surface_data_cur CURSOR FOR
    SELECT s.* from surface_data s, appear_to_surface_data ats
      WHERE s.id=ats.surface_data_id and ats.appearance_id=appearance_row_id;
BEGIN
-- delete surface data not being referenced by appearances any more
  FOR rec IN surface_data_cur LOOP
    IF geodb_pkg.del_is_not_referenced('appear_to_surface_data', 'surface_data_id',
      rec.id, 'appearance_id', appearance_row_id) THEN
        PERFORM geodb_pkg.del_delete_row_surface_data(rec.id);
    END IF;
  END LOOP;

  EXECUTE 'DELETE FROM appear_to_surface_data WHERE appearance_id=$1'
             USING appearance_row_id;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'pre_delete_appearance (id: %): %', appearance_row_id, SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

DELETE.sql (Fortführung)

```
procedure delete_appearance(appearance_rec appearance%rowtype)
is
begin
  pre_delete_appearance(appearance_rec);
  execute immediate 'delete from appearance where id=:1' using appearance_rec.id;
exception
  when others then
    dbms_output.put_line('delete_appearance (id: ' || appearance_rec.id ||
      '): ' || SQLERRM);
end;


/* internal: delete from SURFACE_DATA */
procedure pre_delete_surface_data(surface_data_rec surface_data%rowtype)
is
begin
    execute immediate 'delete from appear_to_surface_data where
                      surface_data_id=:1' using surface_data_rec.id;
    execute immediate 'delete from textureparam where surface_data_id=:1'
                      using surface_data_rec.id;
exception
  when others then
    dbms_output.put_line('pre_delete_surface_data (id: ' || surface_data_rec.id ||
      '): ' || SQLERRM);
end;


procedure delete_surface_data(surface_data_rec surface_data%rowtype)
is
begin
  pre_delete_surface_data(surface_data_rec);
  execute immediate 'delete from surface_data where id=:1'
                    using surface_data_rec.id;
exception
  when others then
    dbms_output.put_line('delete_surface_data (id: ' || surface_data_rec.id ||
      '): ' || SQLERRM);
end;


/* internal: delete from CITYOBJECTGROUP */
procedure pre_delete_cityobjectgroup(cityobjectgroup_rec cityobjectgroup%rowtype)

is
begin
  execute immediate 'delete from group_to_cityobject where cityobjectgroup_id=:1'
                    using cityobjectgroup_rec.id;

exception
  when others then
    dbms_output.put_line('pre_delete_cityobjectgroup (id: ' ||
      cityobjectgroup_rec.id || '): ' || SQLERRM);
end;
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.del_delete_row_appearance(appearance_id
NUMERIC) RETURNS SETOF void AS $$
BEGIN
  PERFORM geodb_pkg.del_pre_delete_appearance(appearance_id);
  EXECUTE 'DELETE FROM appearance WHERE id=$1' USING appearance_id;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_row_appearance (id: %): %', appearance_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


/* internal: DELETE FROM SURFACE_DATA */
CREATE OR REPLACE FUNCTION geodb_pkg.del_pre_delete_surface_data(surface_data_id
NUMERIC) RETURNS SETOF void AS $$
BEGIN
  EXECUTE 'DELETE FROM appear_to_surface_data WHERE surface_data_id=$1'
           USING surface_data_id;
  EXECUTE 'DELETE FROM textureparam WHERE surface_data_id=$1'
           USING surface_data_id;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'pre_delete_surface_data (id: %): %', surface_data_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_delete_row_surface_data(surface_data_id
NUMERIC) RETURNS SETOF void AS $$
BEGIN
  PERFORM geodb_pkg.del_pre_delete_surface_data(surface_data_id);
  EXECUTE 'DELETE FROM surface_data WHERE id=$1' USING surface_data_id;


  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_row_surface_data (id: %): %', surface_data_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


/* internal: DELETE FROM CITYOBJECTGROUP */
CREATE OR REPLACE FUNCTION
geodb_pkg.del_pre_delete_cityobjectgroup(cityobjectgroup_id NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  EXECUTE 'DELETE FROM group_to_cityobject WHERE cityobjectgroup_id=$1'
           USING cityobjectgroup_id;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'pre_delete_cityobjectgroup (id: %): %', cityobjectgroup_id,
                SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

DELETE.sql (Fortführung)

```
procedure delete_cityobjectgroup(cityobjectgroup_rec cityobjectgroup%rowtype)

is

begin
  -- Für die PostgreSQL / PostGIS-Version wurde weitestgehend versucht nur mit ID-
  -- Werten zu arbeiten, da Cursor-Variablen nicht übergeben werden können. Wenn
  -- Funktionen mehrere Werte aus einer Zeile benötigten, musste eine ROWTYPE
  -- Variable befüllt und die aufgerufenen Funktionen spezifisch erweitert werden.
  pre_delete_cityobjectgroup(cityobjectgroup_rec);
  execute immediate 'delete from cityobjectgroup where id=:1'
                    using cityobjectgroup_rec.id;
  post_delete_cityobjectgroup(cityobjectgroup_rec);
exception
  when others then
    dbms_output.put_line('delete_cityobjectgroup (id: ' || cityobjectgroup_rec.id
      || '): ' || SQLERRM);
end;


procedure post_delete_cityobjectgroup(cityobjectgroup_rec cityobjectgroup%rowtype)
-- diese Funktion musste bei der Portierung erweitert werden, siehe rechts
is
begin
  if cityobjectgroup_rec.surface_geometry_id is not null then
    intern_delete_surface_geometry(cityobjectgroup_rec.surface_geometry_id);
  end if;
    intern_delete_cityobject(cityobjectgroup_rec.id);
exception
  when others then
    dbms_output.put_line('post_delete_cityobjectgroup (id: ' ||
      cityobjectgroup_rec.id || '): ' || SQLERRM);
end;


/* internal: delete from THEMATIC_SURFACE */
procedure pre_delete_thematic_surface(thematic_surface_rec thematic_surface
  %rowtype)
is
  cursor opening_cur is
    select o.* from opening o, opening_to_them_surface otm
      where o.id=otm.opening_id and otm.thematic_surface_id =
        thematic_surface_rec.id;
begin
  -- delete openings not being referenced by a thematic surface any more
  for rec in opening_cur loop
    if is_not_referenced('opening_to_them_surface', 'opening_id', rec.id,
      'thematic_surface_id', thematic_surface_rec.id) then
        delete_opening(rec);
    end if;
  end loop;
  execute immediate 'delete from opening_to_them_surface where
                    thematic_surface_id=:1' using thematic_surface_rec.id;
exception
  when others then
    dbms_output.put_line('pre_delete_thematic_surface (id: ' ||
      thematic_surface_rec.id || '): ' || SQLERRM);
end;
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION
geodb_pkg.del_delete_row_cityobjectgroup(cityobjectgroup_id NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  cityobjectgroup_rec cityobjectgroup%ROWTYPE;
BEGIN
  EXECUTE 'SELECT * FROM cityobjectgroup WHERE id=$1'
             INTO cityobjectgroup_rec USING cityobjectgroup_id;

  PERFORM geodb_pkg.del_pre_delete_cityobjectgroup(cityobjectgroup_id);
  EXECUTE 'DELETE FROM cityobjectgroup WHERE id=$1' USING cityobjectgroup_id;
  PERFORM geodb_pkg.del_post_delete_cityobjectgroup(cityobjectgroup_id,
            cityobjectgroup_rec.surface_geometry_id); -- Beispiel für Erweiterung
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_row_cityobjectgroup (id: %): %', cityobjectgroup_id,
                        SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_post_delete_cityobjectgroup(
  cityobjectgroup_id NUMERIC, surface_geometry_id NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  IF surface_geometry_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(surface_geometry_id);
  END IF;
  PERFORM geodb_pkg.del_intern_delete_cityobject(cityobjectgroup_id);
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'post_delete_cityobjectgroup (id: %): %', cityobjectgroup_id,
                        SQLERRM;
END; $$ LANGUAGE plpgsql;


/* internal: DELETE FROM THEMATIC_SURFACE */
CREATE OR REPLACE FUNCTION
geodb_pkg.del_pre_delete_thematic_surface(thematic_surface_row_id NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  opening_cur CURSOR FOR
    SELECT o.* FROM opening o, opening_to_them_surface otm
      WHERE o.id=otm.opening_id AND otm.thematic_surface_id=thematic_surface_row_id;
BEGIN
  -- delete openings not being referenced by a thematic surface any more
  FOR rec IN opening_cur LOOP
    IF geodb_pkg.del_is_not_referenced('opening_to_them_surface', 'opening_id',
      rec.id, 'thematic_surface_id', thematic_surface_row_id) THEN
        PERFORM geodb_pkg.del_delete_row_opening(rec.id);
    END IF;
  END LOOP;
  EXECUTE 'DELETE FROM opening_to_them_surface WHERE thematic_surface_id=$1'
             USING thematic_surface_row_id;
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'pre_delete_thematic_surface (id: %): %',
                        thematic_surface_row_id, SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

DELETE.sql (Fortführung)

```
procedure delete_thematic_surface(thematic_surface_rec thematic_surface%rowtype)

is


begin


  pre_delete_thematic_surface(thematic_surface_rec);
  execute immediate 'delete from thematic_surface where id=:1'
                    using thematic_surface_rec.id;
  post_delete_thematic_surface(thematic_surface_rec);


exception
  when others then
    dbms_output.put_line('delete_thematic_surface (id: ' ||
      thematic_surface_rec.id || '): ' || SQLERRM);
end;


procedure post_delete_thematic_surface(thematic_surface_rec thematic_surface
  %rowtype)

is
begin
  if thematic_surface_rec.lod2_multi_surface_id is not null then
    intern_delete_surface_geometry(thematic_surface_rec.lod2_multi_surface_id);
  end if;
  if thematic_surface_rec.lod3_multi_surface_id is not null then
    intern_delete_surface_geometry(thematic_surface_rec.lod3_multi_surface_id);
  end if;
  if thematic_surface_rec.lod4_multi_surface_id is not null then
    intern_delete_surface_geometry(thematic_surface_rec.lod4_multi_surface_id);
  end if;

  intern_delete_cityobject(thematic_surface_rec.id);
exception
  when others then
    dbms_output.put_line('post_delete_thematic_surface (id: ' ||
      thematic_surface_rec.id || '): ' || SQLERRM);
end;


/* internal: delete from OPENING */
procedure pre_delete_opening(opening_rec opening%rowtype)
is
begin
  execute immediate 'delete from opening_to_them_surface where opening_id=:1'
                    using opening_rec.id;
exception
  when others then
    dbms_output.put_line('pre_delete_opening (id: ' || opening_rec.id || '): ' ||
      SQLERRM);
end;
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION
geodb_pkg.del_delete_row_thematic_surface(thematic_surface_id NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  thematic_surface_rec thematic_surface%ROWTYPE;
BEGIN
  EXECUTE 'SELECT * FROM thematic_surface WHERE id=$1' INTO thematic_surface_rec
             USING thematic_surface_id;
  PERFORM geodb_pkg.del_pre_delete_thematic_surface(thematic_surface_id);
  EXECUTE 'DELETE FROM thematic_surface WHERE id=$1' USING thematic_surface_id;
  PERFORM geodb_pkg.del_post_delete_thematic_surface(thematic_surface_id,
            thematic_surface_rec.lod2_multi_surface_id,
            thematic_surface_rec.lod3_multi_surface_id,
            thematic_surface_rec.lod4_multi_surface_id);
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_row_thematic_surface (id: %): %', thematic_surface_id,
                      SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_post_delete_thematic_surface(
  thematic_surface_id NUMERIC, ts_lod2_multi_surface_id NUMERIC,
  ts_lod3_multi_surface_id NUMERIC, ts_lod4_multi_surface_id NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  IF ts_lod2_multi_surface_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(ts_lod2_multi_surface_id);
  END IF;
  IF ts_lod3_multi_surface_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(ts_lod3_multi_surface_id);
  END IF;
  IF ts_lod4_multi_surface_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(ts_lod4_multi_surface_id);
  END IF;

  PERFORM geodb_pkg.del_intern_delete_cityobject(thematic_surface_id);
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'post_delete_thematic_surface (id: %): %', thematic_surface_id,
                      SQLERRM;
END; $$ LANGUAGE plpgsql;


/* internal: DELETE FROM OPENING */
CREATE OR REPLACE FUNCTION geodb_pkg.del_pre_delete_opening(opening_id NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  EXECUTE 'DELETE FROM opening_to_them_surface WHERE opening_id=$1'
             USING opening_id;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'pre_delete_opening (id: %): %', opening_id, SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

DELETE.sql (Fortführung)

```
procedure delete_opening(opening_rec opening%rowtype)

is

begin

  pre_delete_opening(opening_rec);
  execute immediate 'delete from opening where id=:1' using opening_rec.id;
  post_delete_opening(opening_rec);

exception
  when others then
    dbms_output.put_line('delete_opening (id: ' || opening_rec.id || '): ' ||
      SQLERRM);
end;


procedure post_delete_opening(opening_rec opening%rowtype)


is
  cursor address_cur is
    select a.id from address a left outer join address_to_building ab
      on a.id=ab.address_id where a.id=opening_rec.address_id and ab.address_id
        is null;
begin
  if opening_rec.lod3_multi_surface_id is not null then
    intern_delete_surface_geometry(opening_rec.lod3_multi_surface_id);
  end if;

  if opening_rec.lod4_multi_surface_id is not null then
    intern_delete_surface_geometry(opening_rec.lod4_multi_surface_id);
  end if;


  -- delete addresses not being referenced from buildings and openings any more
  for rec in address_cur loop
    if is_not_referenced('opening', 'address_id', rec.id, 'id', opening_rec.id)
      then
        delete_address(rec.id);
    end if;
  end loop;

  intern_delete_cityobject(opening_rec.id);
exception
  when others then
    dbms_output.put_line('post_delete_opening (id: ' || opening_rec.id || '): '
      || SQLERRM);
end;
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.del_delete_row_opening(opening_id NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  opening_rec opening%ROWTYPE;
BEGIN
  EXECUTE 'SELECT * FROM opening WHERE id=$1' INTO opening_rec USING opening_id;
  PERFORM geodb_pkg.del_pre_delete_opening(opening_id);
  EXECUTE 'DELETE FROM opening WHERE id=$1' USING opening_id;
  PERFORM geodb_pkg.del_post_delete_opening(opening_id, opening_rec.address_id,
            opening_rec.lod3_multi_surface_id, opening_rec.lod4_multi_surface_id);

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_row_opening (id: %): %', opening_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_post_delete_opening(
  opening_id NUMERIC, opening_address_id NUMERIC,
  opening_lod3_multi_surface_id NUMERIC, opening_lod4_multi_surface_id NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  address_cur CURSOR FOR
    SELECT a.id FROM address a LEFT OUTER JOIN address_to_building ab
      ON a.id=ab.address_id WHERE a.id=opening_address_id AND ab.address_id IS NULL;
BEGIN
  IF opening_lod3_multi_surface_id IS NOT NULL THEN
    PERFORM
      geodb_pkg.del_intern_delete_surface_geometry(opening_lod3_multi_surface_id);
  END IF;
  IF opening_lod4_multi_surface_id IS NOT NULL THEN
    PERFORM
      geodb_pkg.del_intern_delete_surface_geometry(opening_lod4_multi_surface_id);
  END IF;

  -- delete addresses not being referenced from buildings and openings any more
  FOR rec IN address_cur LOOP
    IF geodb_pkg.del_is_not_referenced('opening', 'address_id', rec.id, 'id',
      opening_id) THEN
        PERFORM geodb_pkg.del_delete_address(rec.id);
    END IF;
  END LOOP;

  PERFORM geodb_pkg.del_intern_delete_cityobject(opening_id);

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'post_delete_opening (id: %): %', opening_id, SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

DELETE.sql (Fortführung)

```
/* internal: delete from BUILDING_INSTALLATION */
procedure delete_building_installation(building_installation_rec
  building_installation%rowtype)

is

begin


  execute immediate 'delete from building_installation where id=:1'
                      using building_installation_rec.id;
  post_delete_building_inst(building_installation_rec);



exception
  when others then
    dbms_output.put_line('delete_building_installation (id: ' ||
      building_installation_rec.id || '): ' || SQLERRM);
end;


procedure post_delete_building_inst(building_installation_rec
  building_installation%rowtype)

is
begin
  if building_installation_rec.lod2_geometry_id is not null then
    intern_delete_surface_geometry(building_installation_rec.lod2_geometry_id);
  end if;
  if building_installation_rec.lod3_geometry_id is not null then
    intern_delete_surface_geometry(building_installation_rec.lod3_geometry_id);
  end if;
  if building_installation_rec.lod4_geometry_id is not null then
    intern_delete_surface_geometry(building_installation_rec.lod4_geometry_id);
  end if;

  intern_delete_cityobject(building_installation_rec.id);

exception
  when others then
    dbms_output.put_line('post_delete_building_inst (id: ' ||
      building_installation_rec.id || '): ' || SQLERRM);
end;
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
/* internal: DELETE FROM BUILDING_INSTALLATION */
CREATE OR REPLACE FUNCTION
geodb_pkg.del_delete_row_building_inst(building_installation_id NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  building_installation_rec building_installation%ROWTYPE;
BEGIN
  EXECUTE 'SELECT * FROM building_installation WHERE id=$1' INTO
             building_installation_rec USING building_installation_id;
  EXECUTE 'DELETE FROM building_installation WHERE id=$1' USING
             building_installation_id;
  PERFORM geodb_pkg.del_post_delete_building_inst(building_installation_id,
            building_installation_rec.lod2_geometry_id,
            building_installation_rec.lod3_geometry_id,
            building_installation_rec.lod4_geometry_id);
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_row_building_installation (id: %): %',
                    building_installation_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_post_delete_building_inst(
  building_installation_id NUMERIC, bi_lod2_geometry_id NUMERIC,
  bi_lod3_geometry_id NUMERIC, bi_lod4_geometry_id NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  IF bi_lod2_geometry_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(bi_lod2_geometry_id);
  END IF;
  IF bi_lod3_geometry_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(bi_lod3_geometry_id);
  END IF;
  IF bi_lod4_geometry_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(bi_lod4_geometry_id);
  END IF;

  PERFORM geodb_pkg.del_intern_delete_cityobject(building_installation_id);

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'post_delete_building_inst (id: %): %', building_installation_id,
                    SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

DELETE.sql (Fortführung)

```
/* internal: delete from ROOM */
procedure pre_delete_room(room_rec room%rowtype)

is
  cursor thematic_surface_cur is
    select * from thematic_surface where room_id=room_rec.id;
  cursor building_installation_cur is
    select * from building_installation where room_id=room_rec.id;
  cursor building_furniture_cur is
    select * from building_furniture where room_id=room_rec.id;
begin
  for rec in thematic_surface_cur loop
    delete_thematic_surface(rec);
  end loop;
  for rec in building_installation_cur loop
    delete_building_installation(rec);
  end loop;
  for rec in building_furniture_cur loop
    delete_building_furniture(rec);
  end loop;
 exception
  when others then
    dbms_output.put_line('pre_delete_room (id: ' || room_rec.id || '): ' ||
      SQLERRM);
end;


procedure delete_room(room_rec room%rowtype)

is

begin

  pre_delete_room(room_rec);
  execute immediate 'delete from room where id=:1' using room_rec.id;
  post_delete_room(room_rec);
exception
  when others then
    dbms_output.put_line('delete_room (id: ' || room_rec.id || '): ' || SQLERRM);
end;


procedure post_delete_room(room_rec room%rowtype)

is
begin
  if room_rec.lod4_geometry_id is not null then
    intern_delete_surface_geometry(room_rec.lod4_geometry_id);
  end if;

  intern_delete_cityobject(room_rec.id);
exception
  when others then
    dbms_output.put_line('post_delete_room (id: ' || room_rec.id || '): ' ||
      SQLERRM);
end;
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
/* internal: DELETE FROM ROOM */
CREATE OR REPLACE FUNCTION geodb_pkg.del_pre_delete_room(room_row_id NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  thematic_surface_cur CURSOR FOR
    SELECT ts.* FROM thematic_surface ts WHERE ts.room_id=room_row_id;
  building_installation_cur CURSOR FOR
    SELECT bi.* FROM building_installation bi WHERE bi.room_id=room_row_id;
  building_furniture_cur CURSOR FOR
    SELECT bf.* FROM building_furniture bf WHERE bf.room_id=room_row_id;
BEGIN
  FOR rec IN thematic_surface_cur LOOP
    PERFORM geodb_pkg.del_delete_row_thematic_surface(rec.id);
  END LOOP;
  FOR rec IN building_installation_cur LOOP
    PERFORM geodb_pkg.del_delete_row_building_inst(rec.id);
  END LOOP;
  FOR rec IN building_furniture_cur LOOP
    PERFORM geodb_pkg.del_delete_row_building_furniture(rec.id);
  END LOOP;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'pre_delete_room (id: %): %', room_row_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_delete_row_room(room_id NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  room_rec room%ROWTYPE;
BEGIN
  EXECUTE 'SELECT * FROM room WHERE id=$1' INTO room_rec USING room_id;
  PERFORM geodb_pkg.del_pre_delete_room(room_id);
  EXECUTE 'DELETE FROM room WHERE id=$1' USING room_id;
  PERFORM geodb_pkg.del_post_delete_room(room_id, room_rec.lod4_geometry_id);
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_row_room (id: %): %', room_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_post_delete_room(
  room_id NUMERIC, room_lod4_geometry_id NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  IF room_lod4_geometry_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(room_lod4_geometry_id);
  END IF;

  PERFORM geodb_pkg.del_intern_delete_cityobject(room_id);

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'post_delete_room (id: %): %', room_id, SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

DELETE.sql (Fortführung)

```
/* internal: delete from BUILDING_FURNITURE */
procedure delete_building_furniture(building_furniture_rec building_furniture
  %rowtype)

is

begin

  execute immediate 'delete from building_furniture where id=:1'
                     using building_furniture_rec.id;
  post_delete_building_furniture(building_furniture_rec);


exception
  when others then
    dbms_output.put_line('delete_building_furniture (id: ' ||
      building_furniture_rec.id || '): ' || SQLERRM);
end;


procedure post_delete_building_furniture(building_furniture_rec
  building_furniture%rowtype)

is
begin
  if building_furniture_rec.lod4_geometry_id is not null then
    intern_delete_surface_geometry(building_furniture_rec.lod4_geometry_id);
  end if;
  if building_furniture_rec.lod4_implicit_rep_id is not null then
    intern_delete_implicit_geom(building_furniture_rec.lod4_implicit_rep_id);
  end if;

  intern_delete_cityobject(building_furniture_rec.id);

exception
  when others then
    dbms_output.put_line('post_delete_building_furniture (id: ' ||
      building_furniture_rec.id || '): ' || SQLERRM);
end;


/* internal: delete from BUILDING */
procedure pre_delete_building(building_rec building%rowtype)

is
  cursor building_part_cur is
    select * from building where id!=building_rec.id and
      building_parent_id=building_rec.id;
  cursor thematic_surface_cur is
    select * from thematic_surface where building_id=building_rec.id;
  cursor building_installation_cur is
    select * from building_installation where building_id=building_rec.id;
  cursor room_cur is
    select * from room where building_id=building_rec.id;
  cursor address_cur is
    select address_id from address_to_building where building_id=building_rec.id;
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
/* internal: DELETE FROM BUILDING_FURNITURE */
CREATE OR REPLACE FUNCTION
geodb_pkg.del_delete_row_building_furniture(building_furniture_id NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  building_furniture_rec building_furniture%ROWTYPE;
BEGIN
  EXECUTE 'SELECT * FROM building_furniture WHERE id=$1' INTO building_furniture_rec
           USING building_furniture_id;
  EXECUTE 'DELETE FROM building_furniture WHERE id=$1' USING building_furniture_id;
  PERFORM geodb_pkg.del_post_delete_building_furniture(building_furniture_id,
          building_furniture_rec.lod4_geometry_id,
          building_furniture_rec.lod4_implicit_rep_id);
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_row_building_furniture (id: %): %',
                building_furniture_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_post_delete_building_furniture(
  building_furniture_id NUMERIC, bf_lod4_geometry_id NUMERIC,
  bf_lod4_implicit_rep_id NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  IF bf_lod4_geometry_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(bf_rec.lod4_geometry_id);
  END IF;
  IF bf_lod4_implicit_rep_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_implicit_geom(bf_lod4_implicit_rep_id);
  END IF;

  PERFORM geodb_pkg.del_intern_delete_cityobject(building_furniture_id);

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'post_delete_building_furniture (id: %): %',
                building_furniture_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


/* internal: DELETE FROM BUILDING */
CREATE OR REPLACE FUNCTION geodb_pkg.del_pre_delete_building(building_row_id
NUMERIC) RETURNS SETOF void AS $$
DECLARE
  building_part_cur CURSOR FOR
    SELECT * FROM building WHERE id != building_row_id AND
      building_parent_id=building_row_id;
  thematic_surface_cur CURSOR FOR
    SELECT * FROM thematic_surface WHERE building_id=building_row_id;
  building_installation_cur CURSOR FOR
    SELECT * FROM building_installation bi WHERE building_id=building_row_id;
  room_cur CURSOR FOR
    SELECT * FROM room WHERE building_id=building_row_id;
  address_cur CURSOR FOR
    SELECT address_id FROM address_to_building WHERE building_id=building_row_id;
```

Oracle

DELETE.sql (Fortführung)

```
begin
  for rec in building_part_cur loop
    delete_building(rec);
  end loop;
  for rec in thematic_surface_cur loop
    delete_thematic_surface(rec);
  end loop;
  for rec in building_installation_cur loop
    delete_building_installation(rec);
  end loop;
  for rec in room_cur loop
    delete_room(rec);
  end loop;

  -- delete addresses being not referenced from buildings any more
  for rec in address_cur loop
    if is_not_referenced('address_to_building', 'address_id', rec.address_id,
      'building_id', building_rec.id) then
        delete_address(rec.address_id);
    end if;
  end loop;

  execute immediate 'delete from address_to_building where building_id=:1'
                         using building_rec.id;
exception
  when others then
    dbms_output.put_line('pre_delete_building (id: ' || building_rec.id || '): '
      || SQLERRM);
end;


procedure delete_building(building_rec building%rowtype)

is

begin

  pre_delete_building(building_rec);
  execute immediate 'delete from building where id=:1' using building_rec.id;
  post_delete_building(building_rec);


exception
  when others then
    dbms_output.put_line('delete_building (id: ' || building_rec.id || '): ' ||
      SQLERRM);
end;
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
BEGIN
  FOR rec IN building_part_cur LOOP
    PERFORM geodb_pkg.del_delete_row_building(rec.id);
  END LOOP;
  FOR rec IN thematic_surface_cur LOOP
    PERFORM geodb_pkg.del_delete_row_thematic_surface(rec.id);
  END LOOP;
  FOR rec IN building_installation_cur LOOP
    PERFORM geodb_pkg.del_delete_row_building_inst(rec.id);
  END LOOP;
  FOR rec IN room_cur LOOP
    PERFORM geodb_pkg.del_delete_row_room(rec.id);
  END LOOP;

  -- delete addresses being not referenced from buildings any more
  FOR rec IN address_cur LOOP
    IF geodb_pkg.del_is_not_referenced('address_to_building', 'address_id',
      rec.address_id, 'building_id', building_row_id) THEN
        PERFORM geodb_pkg.del_delete_address(rec.address_id);
    END IF;
  END LOOP;

  EXECUTE 'DELETE FROM address_to_building WHERE building_id=$1'
           USING building_row_id;
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'pre_delete_building (id: %): %', building_row_id, SQLERRM;

END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_delete_row_building(building_id NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  building_rec building%ROWTYPE;
BEGIN
  EXECUTE 'SELECT * FROM building WHERE id=$1' INTO building_rec USING building_id;
  PERFORM geodb_pkg.del_pre_delete_building(building_id);
  EXECUTE 'DELETE FROM building WHERE id=$1' USING building_id;
  PERFORM geodb_pkg.del_post_delete_building(building_id,
            building_rec.lod1_geometry_id, building_rec.lod2_geometry_id,
            building_rec.lod3_geometry_id, building_rec.lod4_geometry_id);
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_row_building (id: %): %', building_id, SQLERRM;

END; $$ LANGUAGE plpgsql;
```

Oracle

DELETE.sql (Fortführung)

```
procedure post_delete_building(building_rec building%rowtype)


is
begin
  if building_rec.lod1_geometry_id is not null then
    intern_delete_surface_geometry(building_rec.lod1_geometry_id);
  end if;
  if building_rec.lod2_geometry_id is not null then
    intern_delete_surface_geometry(building_rec.lod2_geometry_id);
  end if;
  if building_rec.lod3_geometry_id is not null then
    intern_delete_surface_geometry(building_rec.lod3_geometry_id);
  end if;
  if building_rec.lod4_geometry_id is not null then
    intern_delete_surface_geometry(building_rec.lod4_geometry_id);
  end if;

  intern_delete_cityobject(building_rec.id);
exception
  when others then
    dbms_output.put_line('post_delete_building (id: ' || building_rec.id || '): '
      || SQLERRM);
end;


/* PUBLIC API PROCEDURES */
procedure delete_surface_geometry(pid number, clean_apps int := 0)

is
begin
  intern_delete_surface_geometry(pid);

  if clean_apps <> 0 then
    cleanup_appearances(0);
  end if;
exception
  when no_data_found then
    return;
  when others then
    dbms_output.put_line('delete_surface_geometry (id: ' || pid || '): ' ||
      SQLERRM);
end;

-- die folgende Prozedur steht prototypisch für andere PUBLIC API PROCEDURES
procedure delete_implicit_geometry(pid number)
is
begin
  intern_delete_implicit_geom(pid);
exception
  when no_data_found then
    return;
  when others then
    dbms_output.put_line('delete_implicit_geometry (id: ' || pid || '): ' ||
      SQLERRM);
end;
. . .
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.del_post_delete_building(
  building_id NUMERIC, b_lod1_geometry_id NUMERIC, b_lod2_geometry_id NUMERIC,
  b_lod3_geometry_id NUMERIC, b_lod4_geometry_id NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  IF b_lod1_geometry_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(b_lod1_geometry_id);
  END IF;
  IF b_lod2_geometry_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(b_lod2_geometry_id);
  END IF;
  IF b_lod3_geometry_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(b_lod3_geometry_id);
  END IF;
  IF b_lod4_geometry_id IS NOT NULL THEN
    PERFORM geodb_pkg.del_intern_delete_surface_geometry(b_lod4_geometry_id);
  END IF;

  PERFORM geodb_pkg.del_intern_delete_cityobject(building_id);

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'post_delete_building (id: %): %', building_id, SQLERRM;
END; $$ LANGUAGE plpgsql;


/* PUBLIC API FUNCTIONS */
CREATE OR REPLACE FUNCTION geodb_pkg.del_delete_surface_geometry(pid NUMERIC,
clean_apps INTEGER DEFAULT 0)
RETURNS SETOF void AS $$
BEGIN
  PERFORM geodb_pkg.del_intern_delete_surface_geometry(pid);

  IF clean_apps <> 0 THEN
    PERFORM geodb_pkg.del_cleanup_appearances(0);
  END IF;

  EXCEPTION
    WHEN no_data_found THEN
      RETURN;
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_surface_geometry (id: %): %', pid, SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_delete_implicit_geometry(pid NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  PERFORM geodb_pkg.del_intern_delete_implicit_geom(pid);

  EXCEPTION
    WHEN no_data_found THEN
      RETURN;
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_implicit_geometry (id: %): %', pid, SQLERRM;
END; $$ LANGUAGE plpgsql;
. . .
```

Oracle

DELETE.sql (Fortführung)

```
  procedure cleanup_appearances(only_global int :=1)

  is
    cursor surface_data_global_cur is
      select s.* from surface_data s left outer join textureparam t
        on s.id=t.surface_data_id where t.surface_data_id is null;
    cursor appearance_cur is
      select a.* from appearance a left outer join appear_to_surface_data asd
        on a.id=asd.appearance_id where asd.appearance_id is null;
    cursor appearance_global_cur is
      select a.* from appearance a left outer join appear_to_surface_data asd
        on a.id=asd.appearance_id where a.cityobject_id is null and
          asd.appearance_id is null;
  begin
    for rec in surface_data_global_cur loop
      delete_surface_data(rec);
    end loop;

    -- delete appearances which does not have surface data any more
    if only_global=1 then
      for rec in appearance_global_cur loop
        delete_appearance(rec);
      end loop;
    else
      for rec in appearance_cur loop
        delete_appearance(rec);
      end loop;
    end if;

  exception
    when others then
      dbms_output.put_line('cleanup_appearances: ' || SQLERRM);
  end;


  procedure cleanup_cityobjectgroups

  is
    cursor group_cur is
      select g.* from cityobjectgroup g left outer join group_to_cityobject gtc
        on g.id=gtc.cityobject_id where gtc.cityobject_id is null;
  begin
    for rec in group_cur loop
      delete_cityobjectgroup(rec);
    end loop;

  exception
    when others then
      dbms_output.put_line('cleanup_cityobjectgroups: ' || SQLERRM);
  end;


  procedure cleanup_citymodels -- äquivalent zur Funktion cleanup_cityobjectgroups
  . . .


END geodb_delete;
/
```

PostgreSQL / PostGIS

DELETE.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.del_cleanup_appearances(only_global INTEGER
  DEFAULT 1) RETURNS SETOF void AS $$
DECLARE
  surface_data_global_cur CURSOR FOR
    SELECT s.* FROM surface_data s LEFT OUTER JOIN textureparam t
      ON s.id=t.surface_data_id WHERE t.surface_data_id IS NULL;
  appearance_cur CURSOR FOR
    SELECT a.* FROM appearance a LEFT OUTER JOIN appear_to_surface_data asd
      ON a.id=asd.appearance_id WHERE asd.appearance_id IS NULL;
  appearance_global_cur CURSOR FOR
    SELECT a.* FROM appearance a LEFT OUTER JOIN appear_to_surface_data asd
      ON a.id=asd.appearance_id WHERE a.cityobject_id IS NULL and
        asd.appearance_id IS NULL;
BEGIN
  FOR rec IN surface_data_global_cur LOOP
    PERFORM geodb_pkg.del_delete_row_surface_data(rec.id);
  END LOOP;

  -- delete appearances which does not have surface data any more
  IF only_global=1 THEN
    FOR rec IN appearance_global_cur LOOP
      PERFORM geodb_pkg.del_delete_row_appearance(rec.id);
    END LOOP;
  ELSE
    FOR rec IN appearance_cur LOOP
      PERFORM geodb_pkg.del_delete_row_appearance(rec.id);
    END LOOP;
  END IF;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'cleanup_appearances: %', SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_cleanup_cityobjectgroups()
RETURNS SETOF void AS $$
DECLARE
  group_cur CURSOR FOR
    SELECT g.* FROM cityobjectgroup g LEFT OUTER JOIN group_to_cityobject gtc
      ON g.id=gtc.cityobject_id WHERE gtc.cityobject_id IS NULL;
BEGIN
  FOR rec IN group_cur LOOP
    PERFORM geodb_pkg.del_delete_row_cityobjectgroup(rec.id);
  END LOOP;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'cleanup_cityobjectgroups: %', SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.del_cleanup_citymodels()
RETURNS SETOF void AS $$
. . .
```

Oracle

DELETE_BY_LINEAGE.sql

```
CREATE OR REPLACE PACKAGE geodb_delete_by_lineage
AS
  procedure delete_buildings(lineage_value varchar2);
END geodb_delete_by_lineage;
/

CREATE OR REPLACE PACKAGE BODY GEODB_DELETE_BY_LINEAGE
AS
  procedure delete_buildings(lineage_value varchar2)

  is
    cursor building_cur is
      select b.id from building b, cityobject c where b.id = c.id and c.lineage =
        lineage_value;
  begin
    for building_rec in building_cur loop
      begin
        geodb_delete.delete_building(building_rec.id);

      exception
        when others then
          dbms_output.put_line('delete_buildings: deletion of building with ID ' ||
            building_rec.id || ' threw: ' || SQLERRM);
      end;
    end loop;

    -- cleanup
    geodb_delete.cleanup_appearances;
    geodb_delete.cleanup_cityobjectgroups;
    geodb_delete.cleanup_citymodels;

  exception
    when others then
      dbms_output.put_line('delete_buildings: ' || SQLERRM);
  end;

END geodb_delete_by_lineage;
/
```

PostgreSQL / PostGIS

DELETE_BY_LINEAGE.sql

```
CREATE OR REPLACE FUNCTION geodb_pkg.del_by_lin_delete_buildings(lineage_value
VARCHAR) RETURNS SETOF void AS $$
DECLARE
  building_cur CURSOR FOR
    SELECT b.id FROM building b, cityobject c WHERE b.id = c.id AND c.lineage =
      lineage_value;
BEGIN
  FOR building_rec IN building_cur LOOP
    BEGIN
      PERFORM geodb_pkg.del_delete_building(building_rec.id);

    EXCEPTION
      WHEN OTHERS THEN
        RAISE NOTICE 'delete_buildings: deletion of building with ID % threw %',
                        building_rec.id, SQLERRM;
    END;
  END LOOP;

  -- cleanup
  PERFORM geodb_pkg.del_cleanup_appearances(1);
  PERFORM geodb_pkg.del_cleanup_cityobjectgroups();
  PERFORM geodb_pkg.del_cleanup_citymodels();

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_buildings: %', SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

IDX.sql

```
/******************************************************************
* TYPE INDEX_OBJ
*
* global type to store information relevant to indexes
******************************************************************/
CREATE OR REPLACE TYPE INDEX_OBJ AS OBJECT

  (index_name       VARCHAR2(100),
   table_name       VARCHAR2(100),
   attribute_name   VARCHAR2(100),
   type             NUMBER(1),
   srid             NUMBER,
   is_3d            NUMBER(1, 0),
     STATIC function construct_spatial_3d
     (index_name VARCHAR2, table_name VARCHAR2, attribute_name VARCHAR2, srid
        NUMBER := 0) RETURN INDEX_OBJ,
     STATIC function construct_spatial_2d
     (index_name VARCHAR2, table_name VARCHAR2, attribute_name VARCHAR2, srid
        NUMBER := 0) RETURN INDEX_OBJ,
     STATIC function construct_normal
     (index_name VARCHAR2, table_name VARCHAR2, attribute_name VARCHAR2, srid
        NUMBER := 0) RETURN INDEX_OBJ
  );
/

/******************************************************************
* TYPE BODY INDEX_OBJ
*
* constructors for INDEX_OBJ instances
******************************************************************/
CREATE OR REPLACE TYPE BODY INDEX_OBJ IS
  STATIC FUNCTION construct_spatial_3d
  (index_name VARCHAR2, table_name VARCHAR2, attribute_name VARCHAR2, srid NUMBER
     := 0) RETURN INDEX_OBJ IS
  BEGIN
    return INDEX_OBJ(upper(index_name), upper(table_name), upper(attribute_name),
      1, srid, 1);
  END;

  STATIC FUNCTION construct_spatial_2d
  (index_name VARCHAR2, table_name VARCHAR2, attribute_name VARCHAR2, srid NUMBER
     := 0) RETURN INDEX_OBJ IS
  BEGIN
    return INDEX_OBJ(upper(index_name), upper(table_name), upper(attribute_name),
      1, srid, 0);
  END;

  STATIC FUNCTION construct_normal
  (index_name VARCHAR2, table_name VARCHAR2, attribute_name VARCHAR2, srid NUMBER
     := 0) RETURN INDEX_OBJ IS
  BEGIN
    return INDEX_OBJ(upper(index_name), upper(table_name), upper(attribute_name),
      0, srid, 0);
  END;
END;
/
```

PostgreSQL / PostGIS

IDX.sql

```
/******************************************************************
* TYPE INDEX_OBJ
*
* global type to store information relevant to indexes
******************************************************************/
DROP TYPE IF EXISTS geodb_pkg.INDEX_OBJ CASCADE;
CREATE TYPE geodb_pkg.INDEX_OBJ AS (
    index_name        VARCHAR(100),
    table_name        VARCHAR(100),
    attribute_name    VARCHAR(100),
    type              NUMERIC(1),
    srid              INTEGER,
    is_3d             NUMERIC(1, 0)
);
```

```
/******************************************************************
* constructors for INDEX_OBJ instances
*
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.idx_construct_spatial_3d
    (index_name VARCHAR, table_name VARCHAR, attribute_name VARCHAR, srid INTEGER
     DEFAULT 0) RETURNS geodb_pkg.INDEX_OBJ AS $$
DECLARE
    idx geodb_pkg.INDEX_OBJ;
BEGIN
    idx.index_name := index_name;
    idx.table_name := table_name;
    idx.attribute_name := attribute_name;
    idx.type := 1;
    idx.srid := srid;
    idx.is_3d := 1;

    RETURN idx;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE STRICT;

-- geodb_pkg.idx_construct_spatial_2d & geodb_pkg.idx_construct_normal sind in ihrem
-- Aufbau äquivalent zur Funktion geodb_pkg.idx_construct_spatial_3d und hier nicht
-- aufgeführt. Die Wertzuweisungen können links in der Oracle-Version eingesehen
-- werden.
```

Oracle

IDX.sql (Fortführung)

```
/******************************************************************
* PACKAGE geodb_idx
*
* utility methods for index handling
*******************************************************************/
CREATE OR REPLACE PACKAGE geodb_idx
AS
  TYPE index_table IS TABLE OF INDEX_OBJ;
  FUNCTION index_status(idx INDEX_OBJ) RETURN VARCHAR2;
   . . .
END geodb_idx;
/

CREATE OR REPLACE PACKAGE BODY geodb_idx
AS
  -- Objekte, die nur im Paket gelten. In PostgreSQL so nicht definierbar.
  NORMAL  CONSTANT NUMBER(1) := 0;
  SPATIAL CONSTANT NUMBER(1) := 1;

  INDICES CONSTANT index_table := index_table(
    INDEX_OBJ.construct_spatial_3d('CITYOBJECT_SPX', 'CITYOBJECT', 'ENVELOPE'),
    INDEX_OBJ.construct_spatial_3d('SURFACE_GEOM_SPX', 'SURFACE_GEOMETRY',
                                    'GEOMETRY'),
    INDEX_OBJ.construct_normal('CITYOBJECT_INX', 'CITYOBJECT', 'GMLID,
                                 GMLID_CODESPACE'),
    INDEX_OBJ.construct_normal('SURFACE_GEOMETRY_INX', 'SURFACE_GEOMETRY', 'GMLID,
                                 GMLID_CODESPACE'),
    INDEX_OBJ.construct_normal('APPEARANCE_INX', 'APPEARANCE', 'GMLID,
                                 GMLID_CODESPACE'),
    INDEX_OBJ.construct_normal('SURFACE_DATA_INX', 'SURFACE_DATA', 'GMLID,
                                 GMLID_CODESPACE'));

  /******************************************************************
  * index_status
  *
  * @param idx index to retrieve status from
  * @return VARCHAR2 string represntation of status, may include
  *                  'DROPPED', 'VALID', 'FAILED', 'INVALID'
  *******************************************************************/
  FUNCTION index_status(idx INDEX_OBJ) RETURN VARCHAR2
  IS
    status VARCHAR2(20);
  BEGIN
    IF idx.type = SPATIAL THEN
      execute immediate 'SELECT UPPER(DOMIDX_OPSTATUS) FROM USER_INDEXES WHERE
        INDEX_NAME=:1' into status using idx.index_name;
    ELSE
      execute immediate 'SELECT UPPER(STATUS) FROM USER_INDEXES WHERE
        INDEX_NAME=:1' into status using idx.index_name;
    END IF;

    RETURN status;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN 'DROPPED';
    WHEN others THEN
      RETURN 'INVALID';
  END;
```

PostgreSQL / PostGIS

IDX.sql (Fortführung)

```
/*****************************************************************
* INDEX_TABLE that holds INDEX_OBJ instances
*
*****************************************************************/
DROP TABLE IF EXISTS geodb_pkg.INDEX_TABLE;
CREATE TABLE geodb_pkg.INDEX_TABLE (
       ID                   SERIAL NOT NULL,
       idx_obj              geodb_pkg.INDEX_OBJ
);

/*****************************************************************
* Populate INDEX_TABLE with INDEX_OBJ instances
*
*****************************************************************/
INSERT INTO geodb_pkg.index_table VALUES (1, geodb_pkg.idx_construct_spatial_3d(
  'cityobject_spx', 'cityobject', 'envelope'));
INSERT INTO geodb_pkg.index_table VALUES (2, geodb_pkg.idx_construct_spatial_3d(
  'surface_geom_spx', 'surface_geometry', 'geometry'));
INSERT INTO geodb_pkg.index_table VALUES (3, geodb_pkg.idx_construct_normal(
  'cityobject_inx', 'cityobject', 'gmlid, gmlid_codespace'));
INSERT INTO geodb_pkg.index_table VALUES (4, geodb_pkg.idx_construct_normal(
  'surface_geometry_inx', 'surface_geometry', 'gmlid, gmlid_codespace'));
INSERT INTO geodb_pkg.index_table VALUES (5, geodb_pkg.idx_construct_normal(
  'appearance_inx', 'appearance', 'gmlid, gmlid_codespace'));
INSERT INTO geodb_pkg.index_table VALUES (6, geodb_pkg.idx_construct_normal(
  'surface_data_inx', 'surface_data', 'gmlid, gmlid_codespace'));

/*****************************************************************
* index_status
*
* @param idx index to retrieve status from
* @return VARCHAR string represntation of status, may include
*                  'DROPPED', 'VALID', 'INVALID', 'FAILED'
*****************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.idx_index_status(idx geodb_pkg.INDEX_OBJ)
RETURNS VARCHAR AS $$
DECLARE
    is_valid BOOLEAN;
    status VARCHAR(20);
BEGIN
    EXECUTE 'SELECT DISTINCT pgi.indisvalid FROM pg_index pgi
        JOIN pg_stat_user_indexes pgsui ON pgsui.relid=pgi.indrelid
        JOIN pg_attribute pga ON pga.attrelid=pgi.indexrelid
        WHERE pgsui.indexrelname=$1' INTO is_valid USING idx.index_name;
    IF is_valid is null THEN
        status := 'DROPPED';
    ELSIF is_valid = true THEN
        status := 'VALID';
    ELSE
        status := 'INVALID';
    END IF;
    RETURN status;
EXCEPTION


    WHEN OTHERS THEN
        RETURN 'FAILED';
END; $$ LANGUAGE plpgsql;
```

Oracle

IDX.sql (Fortführung)

```
/******************************************************************
* index_status
*
* @param table_name table_name of index to retrieve status from
* @param column_name column_name of index to retrieve status from
* @return VARCHAR2 string represntation of status, may include
*                  'DROPPED', 'VALID', 'FAILED', 'INVALID'
******************************************************************/
FUNCTION index_status(table_name VARCHAR2, column_name VARCHAR2) RETURN VARCHAR2

IS
  internal_table_name VARCHAR2(100);
  index_type VARCHAR2(35);
  index_name VARCHAR2(35);
  status VARCHAR2(20);
BEGIN
  internal_table_name := table_name;

  IF geodb_util.versioning_table(table_name) = 'ON' THEN
    internal_table_name := table_name || '_LT';
  END IF;

  execute immediate 'SELECT UPPER(INDEX_TYPE), INDEX_NAME FROM USER_INDEXES WHERE
                     INDEX_NAME=
                       (SELECT UPPER(INDEX_NAME) FROM USER_IND_COLUMNS WHERE
                         TABLE_NAME=UPPER(:1) and COLUMN_NAME=UPPER(:2))'
     into index_type, index_name using internal_table_name, column_name;

  IF index_type = 'DOMAIN' THEN
    execute immediate 'SELECT UPPER(DOMIDX_OPSTATUS) FROM USER_INDEXES WHERE
                       INDEX_NAME=:1' into status using index_name;
  ELSE
    execute immediate 'SELECT UPPER(STATUS) FROM USER_INDEXES WHERE
                       INDEX_NAME=:1' into status using index_name;
  END IF;

  RETURN status;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN 'DROPPED';
  WHEN others THEN
    RETURN 'INVALID';
END;
```

PostgreSQL / PostGIS

IDX.sql (Fortführung)

```
/****************************************************************
* index_status
*
* @param table_name table_name of index to retrieve status from
* @param column_name column_name of index to retrieve status from
* @return VARCHAR string representation of status, may include
*                  'DROPPED', 'VALID', 'INVALID', 'FAILED'
****************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.idx_index_status(table_name VARCHAR,
column_name VARCHAR) RETURNS VARCHAR AS $$
DECLARE
    is_valid BOOLEAN;
    status VARCHAR(20);


BEGIN
  -- Die Zeilen in der Oracle-Version sind notwendig für versionierte Datenbanken
  -- und waren für die PostGIS-Version vorerst nicht konvertierbar.



  EXECUTE 'SELECT DISTINCT pgi.indisvalid FROM pg_index pgi
             JOIN pg_stat_user_indexes pgsui ON pgsui.relid=pgi.indrelid
             JOIN pg_attribute pga ON pga.attrelid=pgi.indexrelid
             WHERE pgsui.relname=$1 AND pga.attname=$2'
               INTO is_valid USING lower(table_name), lower(column_name);

    IF is_valid is null THEN
        status := 'DROPPED';
    ELSIF is_valid = true THEN
        status := 'VALID';
    ELSE
        status := 'INVALID';
    END IF;

    RETURN status;

EXCEPTION
    WHEN OTHERS THEN
        RETURN 'FAILED';
END;
$$
LANGUAGE plpgsql;
```

Oracle

IDX.sql (Fortführung)

```
/******************************************************************
* create_spatial_metadata
*
* @param idx index to create metadata for
******************************************************************/

. . .

/******************************************************************
* create_index
*
* @param idx index to create
* @param is_versioned TRUE if database table is version-enabled
* @return VARCHAR2 sql error code, 0 for no errors
******************************************************************/
FUNCTION create_index(idx INDEX_OBJ, is_versioned BOOLEAN, params VARCHAR2 := '')
  RETURN VARCHAR2
IS
  create_ddl VARCHAR2(1000);
  table_name VARCHAR2(100);
  sql_err_code VARCHAR2(20);
BEGIN
  IF index_status(idx) <> 'VALID' THEN
    sql_err_code := drop_index(idx, is_versioned);
    BEGIN
      table_name := idx.table_name;
      IF is_versioned THEN
        dbms_wm.BEGINDDL(idx.table_name);
        table_name := table_name || '_LTS';
      END IF;
      create_ddl := 'CREATE INDEX ' || idx.index_name || ' ON ' || table_name ||
                '(' || idx.attribute_name || ')';
      IF idx.type = SPATIAL THEN
        create_spatial_metadata(idx, is_versioned);
        create_ddl := create_ddl || ' INDEXTYPE is MDSYS.SPATIAL_INDEX';
      END IF;
      IF params <> '' THEN
        create_ddl := create_ddl || ' ' || params;
      END IF;

      execute immediate create_ddl;

      IF is_versioned THEN
        dbms_wm.COMMITDDL(idx.table_name);
      END IF;

    EXCEPTION
      WHEN others THEN
        dbms_output.put_line(SQLERRM);
        IF is_versioned THEN
          dbms_wm.ROLLBACKDDL(idx.table_name);
        END IF;
        RETURN SQLCODE;
    END;
  END IF;

  RETURN '0';
END;
```

PostgreSQL / PostGIS

IDX.sql (Fortführung)

```
-- In der Oracle-Version bewirkt diese Funktion das Löschen und Erstellen von neuen
-- Metadaten in der Tabelle USER_SDO_GEOM_METADATA. In A.2. wurde bereits auf Seite
-- 111 gezeigt, dass dies für PostGIS nicht erforderlich ist. Die Funktion konnte
-- demnach entfallen.



/******************************************************************
* create_index
*
* @param idx index to create
* @param params additional parameters for the index to be created
* @return VARCHAR sql error code and message, 0 for no errors
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.idx_create_index(idx geodb_pkg.INDEX_OBJ,
params VARCHAR DEFAULT '') RETURNS VARCHAR AS $$
DECLARE
    create_ddl VARCHAR(1000);
    SPATIAL CONSTANT NUMERIC(1) := 1;

BEGIN
    IF geodb_pkg.idx_index_status(idx) <> 'VALID' THEN
        PERFORM geodb_pkg.idx_drop_index(idx);
        BEGIN
          -- Die Zeilen in der Oracle-Version sind notwendig für versionierte
          -- Datenbanken und waren für die PostGIS-Version vorerst nicht
          -- konvertierbar.

          IF idx.type = SPATIAL THEN
            create_ddl := 'CREATE INDEX ' || idx.index_name || ' ON ' ||
                             idx.table_name || ' USING GIST (' || idx.attribute_name
                             || ' gist_geometry_ops_nd)';
          ELSE
            create_ddl := 'CREATE INDEX ' || idx.index_name || ' ON ' ||
                             idx.table_name || '(' || idx.attribute_name || ')';
          END IF;

          IF params <> '' THEN
              create_ddl := create_ddl || ' ' || params;
          END IF;

          EXECUTE create_ddl;

        EXCEPTION
            WHEN OTHERS THEN
                RETURN SQLSTATE || ' - ' || SQLERRM;
        END;
    END IF;

    RETURN '0';
END;
$$
LANGUAGE plpgsql;
```

Oracle

IDX.sql (Fortführung)

```
/******************************************************************
* drop_index
*
* @param idx index to drop
* @param is_versioned TRUE if database table is version-enabled
* @return VARCHAR2 sql error code, 0 for no errors
******************************************************************/
FUNCTION drop_index(idx INDEX_OBJ, is_versioned BOOLEAN) RETURN VARCHAR2
IS
  index_name VARCHAR2(100);
BEGIN
  IF index_status(idx) <> 'DROPPED' THEN
    BEGIN
      index_name := idx.index_name;
      IF is_versioned THEN
        dbms_wm.BEGINDDL(idx.table_name);
        index_name := index_name || '_LTS';
      END IF;

      execute immediate 'DROP INDEX ' || index_name;

      IF is_versioned THEN
        dbms_wm.COMMITDDL(idx.table_name);
      END IF;
    EXCEPTION
      WHEN others THEN
        dbms_output.put_line(SQLERRM);
        IF is_versioned THEN
          dbms_wm.ROLLBACKDDL(idx.table_name);
        END IF;

        RETURN SQLCODE;
    END;
  END IF;

  RETURN '0';
END;

/******************************************************************
* create_indexes
* private convience method for invoking create_index on indexes
* of same index type
*
* @param type type of index, e.g. SPATIAL or NORMAL
* @return STRARRAY array of log message strings
******************************************************************/
FUNCTION create_indexes(type SMALLINT) RETURN STRARRAY

IS
  log STRARRAY;
  sql_error_code VARCHAR2(20);
BEGIN
  log := STRARRAY();
  FOR i IN INDICES.FIRST .. INDICES.LAST LOOP
    IF INDICES(i).type = type THEN
      sql_error_code := create_index(INDICES(i),
                                geodb_util.versioning_table(INDICES(i).table_name) =
                                  'ON');
```

PostgreSQL / PostGIS

IDX.sql (Fortführung)

```
/*****************************************************************
* drop_index
*
* @param idx index to drop
* @param is_versioned TRUE IF database table is version-enabled
* @return VARCHAR sql error code and message, 0 for no errors
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.idx_drop_index(idx geodb_pkg.INDEX_OBJ)
RETURNS VARCHAR AS $$
DECLARE
    index_name VARCHAR(100);
BEGIN
    IF geodb_pkg.idx_index_status(idx) <> 'DROPPED' THEN
        BEGIN
          -- Die Zeilen in der Oracle-Version sind notwendig für versionierte
          -- Datenbanken und waren für die PostGIS-Version vorerst nicht
          -- konvertierbar.


          EXECUTE 'DROP INDEX IF EXISTS ' || idx.index_name;




          EXCEPTION
            WHEN OTHERS THEN





              RETURN SQLSTATE || ' - ' || SQLERRM;
     END;
   END IF;

   RETURN '0';
END; $$ LANGUAGE plpgsql;

/*****************************************************************
* create_indexes
* private convience method for invoking create_index on indexes
* of same index type
*
* @param type type of index, e.g. 1 for spatial, 0 for normal
* @return ARRAY array of log message strings
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.idx_create_indexes(type INTEGER)
RETURNS text[] AS $$
DECLARE
    log text[] := '{}';
    sql_error_msg VARCHAR;
    rec RECORD;
BEGIN
    FOR rec IN select * from geodb_pkg.index_table LOOP
        IF (rec.idx_obj).type = type THEN
            sql_error_msg := geodb_pkg.idx_create_index(rec.idx_obj);
```

Oracle

IDX.sql (Fortführung)

```
        log.extend;
        log(log.count) := index_status(INDICES(i)) || ':' || INDICES(i).index_name
                                    || ':' || INDICES(i).table_name || ':' ||
                                    INDICES(i).attribute_name || ':' || sql_error_code;
      END IF;
    END LOOP;

    RETURN log;
END;

/***************************************************************
* drop_indexes
* private convience method for invoking drop_index on indexes
* of same index type
*
* @param type type of index, e.g. SPATIAL or NORMAL
* @return STRARRAY array of log message strings
****************************************************************/
FUNCTION drop_indexes(type SMALLINT) RETURN STRARRAY
. . . -- ist nahezu identisch mit create_indexes(type SMALLINT)

/***************************************************************
* status_spatial_indexes
*
* @return STRARRAY array of log message strings
****************************************************************/
FUNCTION status_spatial_indexes RETURN STRARRAY

IS
    log STRARRAY;
    status VARCHAR2(20);

BEGIN
    log := STRARRAY();

    FOR i in INDICES.FIRST .. INDICES.LAST LOOP
      IF INDICES(i).type = SPATIAL THEN
        status := index_status(INDICES(i));
        log.extend;
        log(log.count) := status || ':' || INDICES(i).index_name || ':' ||
                                    INDICES(i).table_name || ':' || INDICES(i).attribute_name;
      END IF;
    END LOOP;

    RETURN log;
END;

/***************************************************************
* status_normal_indexes
*
* @return STRARRAY array of log message strings
****************************************************************/
FUNCTION status_normal_indexes RETURN STRARRAY
. . . -- ist nahezu identisch mit status_spatial_indexes RETURN STRARRAY
```

PostgreSQL / PostGIS

IDX.sql (Fortführung)

```
      log := array_append(log, geodb_pkg.idx_index_status(rec.idx_obj) || ':' ||
               (rec.idx_obj).index_name || ':' || (rec.idx_obj).table_name || ':' ||
               (rec.idx_obj).attribute_name || ':' || sql_error_msg);

    END IF;
  END LOOP;

  RETURN log;
END; $$ LANGUAGE plpgsql;

/******************************************************************
* drop_indexes
* private convience method for invoking drop_index on indexes
* of same index type
*
* @param type type of index, e.g. 1 for spatial, 0 for normal
* @return ARRAY array of log message strings
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.idx_drop_indexes(type INTEGER)
RETURNS text[] AS $$ . . . -- siehe links

/******************************************************************
* status_spatial_indexes
*
* @return ARRAY array of log message strings
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.idx_status_spatial_indexes() RETURNS text[] AS
$$
DECLARE
    log text[] := '{}';
    status VARCHAR(20);
    rec RECORD;
BEGIN


    FOR rec IN select * from geodb_pkg.index_table LOOP
        IF (rec.idx_obj).type = 1 THEN
            status := geodb_pkg.idx_index_status(rec.idx_obj);
            log := array_append(log, status || ':' || (rec.idx_obj).index_name ||
            ':' || (rec.idx_obj).table_name || ':' || (rec.idx_obj).attribute_name);
        END IF;
    END LOOP;


   RETURN log;
END; $$ LANGUAGE plpgsql;

/******************************************************************
* status_normal_indexes
*
* @return ARRAY array of log message strings
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.idx_status_normal_indexes()
RETURNS text[] AS $$ . . . -- siehe links
```

Oracle

IDX.sql (Fortführung)

```
  /*****************************************************************
  * create_spatial_indexes
  * convience method for invoking create_index on all spatial
  * indexes
  *
  * @return STRARRAY array of log message strings
  ******************************************************************/
  FUNCTION create_spatial_indexes RETURN STRARRAY
  IS
  BEGIN
    dbms_output.enable;
    return create_indexes(SPATIAL);
  END;
  . . .
  -- drop_spatial_indexes, create_normal_indexes, drop_normal_indexes sind nahezu
  -- identisch.

  /*****************************************************************
  * get_index
  * convience method for getting an index object
  * given the table and column it indexes
  *
  * @param table_name
  * @param attribute_name
  * @return INDEX_OBJ
  ******************************************************************/
  FUNCTION get_index(table_name VARCHAR2, column_name VARCHAR2) RETURN INDEX_OBJ

  IS
    idx INDEX_OBJ;

  BEGIN
    FOR i in INDICES.FIRST .. INDICES.LAST LOOP
      IF INDICES(i).attribute_name = UPPER(column_name) AND INDICES(i).table_name =
        UPPER(table_name) THEN
          idx := INDICES(i);
        EXIT;
      END IF;
    END LOOP;

    RETURN idx;
  END;
END geodb_idx;
/
```

PostgreSQL / PostGIS

IDX.sql (Fortführung)

```
/******************************************************************
* create_spatial_indexes
* convience method for invoking create_index on all spatial
* indexes
*
* @return ARRAY array of log message strings
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.idx_create_spatial_indexes()
RETURNS text[] AS $$
BEGIN
  RETURN geodb_pkg.idx_create_indexes(1);
END; $$ LANGUAGE plpgsql;
. . .
```

```
/******************************************************************
* get_index
* convience method for getting an index object
* given the table and column it indexes
*
* @param table_name
* @param attribute_name
* @return INDEX_OBJ
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.idx_get_index(table_name VARCHAR, column_name
VARCHAR) RETURNS geodb_pkg.INDEX_OBJ AS $$
DECLARE
    idx geodb_pkg.INDEX_OBJ;
    rec RECORD;
BEGIN
    FOR rec IN select * from geodb_pkg.index_table LOOP
        IF (rec.idx_obj).attribute_name = column_name AND (rec.idx_obj).table_name =
          table_name THEN
            idx := rec.idx_obj;
        END IF;
    END LOOP;

    RETURN idx;
END; $$ LANGUAGE plpgsql;
```

Oracle

MATCH.sql

```
set term off;
set serveroutput off;

drop table match_overlap_all;
create table match_overlap_all
      (id1 number,
      parent_id1 number,
      root_id1 number,
      area1 number,
      lod1 number,
      lineage varchar2(256),
      id2 number,
      parent_id2 number,
      root_id2 number,
      area2 number,
      lod2 number,
      intersection_geometry mdsys.sdo_geometry,
      intersection_area number,
      area1_cov_by_area2 number,
      area2_cov_by_area1 number)
      nologging;

drop table match_overlap_relevant;
create table match_overlap_relevant
      (id1 number,
      parent_id1 number,
      root_id1 number,
      area1 number,
      lod1 number,
      lineage varchar2(256),
      id2 number,
      parent_id2 number,
      root_id2 number,
      area2 number,
      lod2 number,
      intersection_geometry mdsys.sdo_geometry,
      intersection_area number,
      area1_cov_by_area2 number,
      area2_cov_by_area1 number)
      nologging;

drop table match_master_projected;
create table match_master_projected
      (id number,
      parent_id number,
      root_id number,
      geometry mdsys.sdo_geometry)
      nologging;

drop table match_cand_projected;
create table match_cand_projected
      (id number,
      parent_id number,
      root_id number,
      geometry mdsys.sdo_geometry)
      nologging;
```

PostgreSQL / PostGIS

MATCH.sql

```
DROP TABLE IF EXISTS geodb_pkg.match_overlap_all;
CREATE TABLE geodb_pkg.match_overlap_all(
  id1 INTEGER,
  parent_id1 INTEGER,
  root_id1 INTEGER,
  area1 NUMERIC,
  lod1 NUMERIC,
  lineage VARCHAR(256),
  id2 INTEGER,
  parent_id2 INTEGER,
  root_id2 INTEGER,
  area2 NUMERIC,
  lod2 NUMERIC,
  intersection_geometry GEOMETRY,
  intersection_area NUMERIC,
  area1_cov_by_area2 NUMERIC,
  area2_cov_by_area1 NUMERIC
);

DROP TABLE IF EXISTS geodb_pkg.match_overlap_relevant;
CREATE TABLE geodb_pkg.match_overlap_relevant(
  id1 INTEGER,
  parent_id1 INTEGER,
  root_id1 INTEGER,
  area1 NUMERIC,
  lod1 NUMERIC,
  lineage VARCHAR(256),
  id2 INTEGER,
  parent_id2 INTEGER,
  root_id2 INTEGER,
  area2 NUMERIC,
  lod2 NUMERIC,
  intersection_geometry GEOMETRY,
  intersection_area NUMERIC,
  area1_cov_by_area2 NUMERIC,
  area2_cov_by_area1 NUMERIC
);

DROP TABLE IF EXISTS geodb_pkg.MATCH_MASTER_PROJECTED;
CREATE TABLE geodb_pkg.match_master_projected(
  id INTEGER,
  parent_id INTEGER,
  root_id INTEGER,
  geometry GEOMETRY
);

DROP TABLE IF EXISTS geodb_pkg.MATCH_CAND_PROJECTED;
CREATE TABLE geodb_pkg.match_cand_projected(
  id INTEGER,
  parent_id INTEGER,
  root_id INTEGER,
  geometry GEOMETRY
);
```

Oracle

MATCH.sql (Fortführung)

```
drop table match_collect_geom;
create table match_collect_geom
       (id number,
       parent_id number,
       root_id number,
       geometry mdsys.sdo_geometry)
       nologging;

truncate table match_tmp_building;
drop table match_tmp_building;
create global temporary table match_tmp_building
       (id number,
       parent_id number,
       root_id number,
       geometry_id number)
       on commit preserve rows;

set term on;
set serveroutput on;

CREATE OR REPLACE PACKAGE geodb_match
AS
  procedure create_matching_table(lod_cand number, lineage cityobject.lineage%type,
    lod_master number, delta_cand number, delta_master number, tolerance number :=
    0.001, aggregate_building number := 1);
  . . .
END geodb_match;
/

CREATE OR REPLACE PACKAGE BODY geodb_match
AS
  -- private procedures
  function get_2d_srid return number;

  -- private constants
  CAND_GEOMETRY_TABLE constant string(30) := 'MATCH_CAND_PROJECTED';
  MASTER_GEOMETRY_TABLE constant string(30) := 'MATCH_MASTER_PROJECTED';

  -- declaration of private indexes
  match_master_projected_spx index_obj :=
    index_obj.construct_spatial_2d('match_master_projected_spx', MASTER_GEOMETRY_TABLE,
      'geometry');
  match_cand_projected_spx index_obj :=
    index_obj.construct_spatial_2d('match_cand_projected_spx', CAND_GEOMETRY_TABLE,
      'geometry');
  match_overlap_all_spx index_obj :=
    index_obj.construct_spatial_2d('match_overlap_all_spx', 'match_overlap_all',
      'intersection_geometry');
  match_result_spx index_obj :=
    index_obj.construct_spatial_2d('match_overlap_relevant_spx',
      'match_overlap_relevant', 'intersection_geometry');
  match_collect_id_idx index_obj :=
    index_obj.construct_normal('match_collect_id_idx', 'match_collect_geom', 'id');
  match_collect_root_id_idx index_obj :=
    index_obj.construct_normal('match_collect_root_id_idx', 'match_collect_geom',
      'root_id');
```

PostgreSQL / PostGIS

MATCH.sql (Fortführung)

```
DROP TABLE IF EXISTS geodb_pkg.match_collect_geom;
CREATE TABLE geodb_pkg.match_collect_geom(
  id INTEGER,
  parent_id INTEGER,
  root_id INTEGER,
  geometry GEOMETRY
);
```

```
-- Konstante, die nur im Paket gilt. In PostgreSQL so nicht definierbar.


-- Objekte, die nur im Paket gelten. In PostgreSQL so nicht definierbar.
```

Oracle

MATCH.sql (Fortführung)

```
procedure create_matching_table(lod_cand number, lineage cityobject.lineage%type,
  lod_master number, delta_cand number, delta_master number, tolerance number :=
  0.001, aggregate_building number := 1)

is
  log varchar2(4000);
begin
  -- gather candidate buildings
  collect_cand_building(lod_cand, lineage);
  -- gather candidate geometry
  collect_geometry(lod_cand);
  -- rectify candidate geometry
  rectify_geometry(tolerance);
  -- aggregate candidate geometry
  aggregate_geometry(CAND_GEOMETRY_TABLE, tolerance, aggregate_building);
  -- gather master buildings

  collect_master_building(lod_master, lineage);
  -- gather master geometry
  collect_geometry(lod_master);
  -- rectify master geometry
  rectify_geometry(tolerance);
  -- aggregate master geometry
  aggregate_geometry(MASTER_GEOMETRY_TABLE, tolerance, aggregate_building);
  -- fill matching table

  join_cand_master(lod_cand, lineage, lod_master, tolerance);
  -- densify to 1:1 matches
  create_relevant_matches(delta_cand, delta_master);

  commit;
end;


procedure collect_cand_building(lod number, lineage cityobject.lineage%type)

is
begin
  -- truncate tmp table
  execute immediate 'truncate table match_tmp_building';

  -- Temporäre Tabellen müssen in PostgreSQL bei jedem Aufbauen einer neuen
  -- Verbindung zur Datenbank („Session") neu erstellt werden. Sie können daher
  -- nicht beim Anlegen der 3DCityDB deklariert werden. Als Lösung wird die
  -- Tabelle match_tmp_building nun in jener Funktion erstellt, die als erste
  -- vom Importer/Exporter aufgerufen wird. Temporäre Tabellen können außerdem
  -- nicht im Schema geodb_pkg abgespeichert werden, weil es sich um ein nicht-
  -- temporäres Schema handelt.

  -- retrieve all building tupels belonging to the specified lineage
  execute immediate 'insert all into match_tmp_building
    select b.id, b.building_parent_id parent_id, b.building_root_id root_id,
      b.lod'||to_char(lod)||'_geometry_id geometry_id
      from building b, cityobject c
      where c.id = b.id
        and c.lineage = :1' using lineage;
end;
```

PostgreSQL / PostGIS

MATCH.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.match_create_matching_table(lod_cand INTEGER,
lineage cityobject.lineage%TYPE, lod_master INTEGER, delta_cand INTEGER,
delta_master INTEGER, aggregate_building NUMERIC DEFAULT 1)
RETURNS SETOF void AS $$
DECLARE
  log VARCHAR(4000);
BEGIN
  -- gather candidate buildings
  PERFORM geodb_pkg.match_collect_cand_building(lod_cand, lineage);
  -- gather candidate geometry
  PERFORM geodb_pkg.match_collect_geometry(lod_cand);
  -- rectify candidate geometry
  PERFORM geodb_pkg.match_rectify_geometry();
  -- aggregate candidate geometry
  PERFORM geodb_pkg.match_aggregate_geometry('geodb_pkg.MATCH_CAND_PROJECTED',
            aggregate_building);
  -- gather master buildings
  PERFORM geodb_pkg.match_collect_master_building(lod_master, lineage);
  -- gather master geometry
  PERFORM geodb_pkg.match_collect_geometry(lod_master);
  -- rectify master geometry
  PERFORM geodb_pkg.match_rectify_geometry();
  -- aggregate master geometry
  PERFORM geodb_pkg.match_aggregate_geometry('geodb_pkg.MATCH_MASTER_PROJECTED',
            aggregate_building);
  -- fill matching table
  PERFORM geodb_pkg.match_join_cand_master(lod_cand, lineage, lod_master);
  -- densify to 1:1 matches
  PERFORM geodb_pkg.match_create_relevant_matches(delta_cand, delta_master);

  COMMIT;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.match_collect_cand_building(lod INTEGER,
lineage cityobject.lineage%TYPE)
RETURNS SETOF void AS $$
BEGIN
  -- creates the temporary table match_tmp_building
  EXECUTE 'CREATE GLOBAL TEMPORARY TABLE match_tmp_building(
    id INTEGER,
    parent_id INTEGER,
    root_id INTEGER,
    geometry_id INTEGER
    ) ON COMMIT PRESERVE ROWS';



-- retrieve all building tupels belonging to the specified lineage
  EXECUTE 'INSERT INTO match_tmp_building
    SELECT b.id, b.building_parent_id parent_id, b.building_root_id root_id,
      b.lod'||lod||'_geometry_id geometry_id
      FROM building b, cityobject c WHERE c.id = b.id
        AND c.lineage = $1' USING lineage;

END; $$ LANGUAGE plpgsql;
```

Oracle

MATCH.sql (Fortführung)

```
procedure collect_master_building(lod number, lineage cityobject.lineage%type)
is
begin
  -- truncate tmp table
  execute immediate 'truncate table match_tmp_building';
  -- retrieve all building tupels not belonging to the specified lineage and
  -- whose mbr is interacting with the aggregated mbr of all candidate building
  -- footprint
  execute immediate 'insert all into match_tmp_building
    select b.id, b.building_parent_id parent_id, b.building_root_id root_id,
      b.lod'||to_char(lod)||'_geometry_id geometry_id from building b, cityobject
       c where c.id = b.id and sdo_filter(c.envelope, :2) = ''TRUE'' and
         (c.lineage <> :1 or c.lineage is null)'
            using aggregate_mbr(CAND_GEOMETRY_TABLE), lineage;
end;


procedure collect_geometry(lod number)
is
  log varchar2(4000);
  srid number;
begin
  -- first, truncate tmp table
  execute immediate 'truncate table match_collect_geom';
  log := geodb_idx.drop_index(match_collect_id_idx, false);
  log := geodb_idx.drop_index(match_collect_root_id_idx, false);
  -- get srid for 2d geometries
  srid := get_2d_srid;
  -- second, retrieve exterior shell surfaces from building
  execute immediate 'insert all /*+ append nologging */ into match_collect_geom
    select bl.id, bl.parent_id, bl.root_id, geodb_util.to_2d(s.geometry, :1)
      from match_tmp_building bl, surface_geometry s
      where s.root_id = bl.geometry_id and s.geometry is not null' using srid;
  -- for lod > 1 we also have to check surfaces from the tables
  -- building_installation and thematic surface
  if lod > 1 then
    -- retrieve surfaces from building installations referencing the identified
    -- building tupels
    execute immediate 'insert all /*+ append nologging */ into match_collect_geom
      select bl.id, bl.parent_id, bl.root_id, geodb_util.to_2d(s.geometry, :1)
        from match_tmp_building bl, building_installation i, surface_geometry s
        where i.building_id = bl.id and i.is_external = 1 and s.root_id =
          i.lod'||to_char(lod)||'_geometry_id and s.geometry is not null'
      using srid;
    -- retrieve surfaces from thematic surfaces referencing the identified
    -- building tupels
    execute immediate 'insert all /*+ append nologging */ into match_collect_geom
      select bl.id, bl.parent_id, bl.root_id, geodb_util.to_2d(s.geometry, :1)
        from match_tmp_building bl, thematic_surface t, surface_geometry s
        where t.building_id = bl.id and upper(t.type) not in
        (''INTERIORWALLSURFACE'', ''CEILINGSURFACE'', ''FLOORSURFACE'') and
        s.root_id = t.lod'||to_char(lod)||'_multi_surface_id and s.geometry is
        not null' using srid;
  end if;
exception
  when others then
    dbms_output.put_line('collect_geometry: ' || SQLERRM);
end;
```

PostgreSQL / PostGIS

MATCH.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.match_collect_master_building(lod INTEGER,
lineage cityobject.lineage%TYPE)
RETURNS SETOF void AS $$
BEGIN
  -- truncate tmp table
  EXECUTE 'TRUNCATE TABLE match_tmp_building';
  -- retrieve all building tupels not belonging to the specified lineage and
  -- whose mbr is interacting with the aggregated mbr of all candidate building
footprint
  EXECUTE 'INSERT INTO match_tmp_building
    SELECT b.id, b.building_parent_id parent_id, b.building_root_id root_id,
      b.lod'||lod||'_geometry_id geometry_id FROM building b, cityobject c
        WHERE c.id = b.id AND ST_Intersects(c.envelope, $1) = ''TRUE'' AND
          (c.lineage != $2 OR c.lineage IS NULL)'
    USING geodb_pkg.match_aggregate_mbr('geodb_pkg.MATCH_CAND_PROJECTED'), lineage;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.match_collect_geometry(lod INTEGER)
RETURNS SETOF void AS $$
DECLARE
  srid  INTEGER;
BEGIN
  -- first, truncate tmp table
  EXECUTE 'TRUNCATE TABLE geodb_pkg.match_collect_geom';
  EXECUTE 'DROP INDEX IF EXISTS match_collect_id_idx';
  EXECUTE 'DROP INDEX IF EXISTS match_collect_root_id_idx';
  -- second, retrieve exterior shell surfaces FROM building
  EXECUTE 'INSERT INTO geodb_pkg.match_collect_geom
    SELECT bl.id, bl.parent_id, bl.root_id, ST_Force_2D(s.geometry)
      FROM match_tmp_building bl, surface_geometry s WHERE s.root_id =
        bl.geometry_id AND s.geometry IS NOT NULL';
  -- for lod > 1 we also have to check surfaces FROM the tables
  -- building_installation and thematic surface
  IF lod > 1 THEN
    -- retrieve surfaces FROM building installations referencing the identified
    -- building tupels
    EXECUTE 'INSERT INTO geodb_pkg.match_collect_geom
      SELECT bl.id, bl.parent_id, bl.root_id, ST_Force_2D(s.geometry)
        FROM match_tmp_building bl, building_installation i, surface_geometry s
          WHERE i.building_id = bl.id AND i.is_external = 1 AND s.root_id = i.lod'
            ||lod||'_geometry_id AND s.geometry IS NOT NULL';
    -- retrieve surfaces FROM thematic surfaces referencing the identified
    -- building tupels
    EXECUTE 'INSERT INTO geodb_pkg.match_collect_geom
         SELECT bl.id, bl.parent_id, bl.root_id, ST_Force_2D(s.geometry)
        FROM match_tmp_building bl, thematic_surface t, surface_geometry s
          WHERE t.building_id = bl.id AND upper(t.type) NOT IN
            (''INTERIORWALLSURFACE'', ''CEILINGSURFACE'', ''FLOORSURFACE'')
            AND s.root_id = t.lod'||lod||'_multi_surface_id
            AND s.geometry IS NOT NULL';
  END IF;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'collect_geometry: %', SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

MATCH.sql (Fortführung)

```
procedure rectify_geometry(tolerance number := 0.001)
is
begin
  -- first, remove invalid geometries
  execute immediate 'delete from match_collect_geom where
    sdo_geom.validate_geometry(geometry, :1) <> ''TRUE''' using tolerance;
  -- second, delete vertical surfaces
  execute immediate 'delete from match_collect_geom where
    sdo_geom.sdo_area(geometry, :1) <= :2' using tolerance, tolerance;
exception
  when others then
    dbms_output.put_line('rectify_geometry: ' || SQLERRM);
end;


procedure aggregate_geometry(table_name varchar2, tolerance number := 0.001,
aggregate_building number := 1)

is
  log varchar2(4000);
  -- In der Oracle-Version verwenden einige Funktionen Variablen, die speziell
  -- für das Package definiert wurden (S.189). In der PostgreSQL/PostGIS-Version
  -- müssen diese Variablen innerhalb der Funktionen definiert werden.

begin









  -- truncate table
  execute immediate 'truncate table '||table_name;
  -- drop spatial indexes
  if match_cand_projected_spx.table_name = table_name then
    log := geodb_idx.drop_index(match_cand_projected_spx, false);
  else
    log := geodb_idx.drop_index(match_master_projected_spx, false);
  end if;

  if aggregate_building > 0 then
    declare
      cursor root_id_cur is
        select distinct root_id from match_tmp_building;

    begin
      log:=geodb_idx.create_index(match_collect_root_id_idx, false, 'nologging');

      for root_id_rec in root_id_cur loop
        execute immediate 'insert into '||table_name||' (id, parent_id, root_id,
          geometry) values (:1, null, :2, :3)'
          using root_id_rec.root_id, root_id_rec.root_id,
                aggregate_geometry_by_id(root_id_rec.root_id, tolerance, 1);
      end loop;
    end;
```

PostgreSQL / PostGIS

MATCH.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.match_rectify_geometry()
RETURNS SETOF void AS $$
BEGIN
  -- first, remove invalid geometries
  EXECUTE 'DELETE FROM geodb_pkg.match_collect_geom WHERE ST_IsValid(geometry) !=
    ''TRUE'''; --USING tolerance; --no tolerance in PostGIS-function
  -- second, DELETE vertical surfaces
  EXECUTE 'DELETE FROM geodb_pkg.match_collect_geom WHERE ST_Area(geometry) <=
  0.001';
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'rectify_geometry: %', SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.match_aggregate_geometry(table_name VARCHAR,
aggregate_building INTEGER DEFAULT 1)
RETURNS SETOF void AS $$
DECLARE
  log VARCHAR(4000);
  match_cand_projected_spx geodb_pkg.index_obj;
  match_master_projected_spx geodb_pkg.index_obj;
  match_collect_id_idx geodb_pkg.index_obj;
  match_collect_root_id_idx geodb_pkg.index_obj;
BEGIN
  match_cand_projected_spx := geodb_pkg.idx_construct_spatial_2d(
    'match_cand_projected_spx', 'geodb_pkg.MATCH_CAND_PROJECTED', 'geometry');
  match_master_projected_spx := geodb_pkg.idx_construct_spatial_2d(
    'match_master_projected_spx', 'geodb_pkg.MATCH_MASTER_PROJECTED', 'geometry');
  match_collect_id_idx := geodb_pkg.idx_construct_normal(
    'match_collect_id_idx', 'geodb_pkg.match_collect_geom', 'id');
  match_collect_root_id_idx := geodb_pkg.idx_construct_normal(
    'match_collect_root_id_idx', 'geodb_pkg.match_collect_geom', 'root_id');

  -- TRUNCATE TABLE
  EXECUTE 'TRUNCATE TABLE '||table_name;
  -- drop spatial indexes
  IF (match_cand_projected_spx).table_name = table_name THEN
    log := geodb_pkg.idx_drop_index(match_cand_projected_spx);
  ELSE
    log := geodb_pkg.idx_drop_index(match_master_projected_spx);
  END IF;

  IF aggregate_building > 0 then
    DECLARE
      root_id_cur CURSOR FOR
        SELECT DISTINCT root_id FROM match_tmp_building;

    BEGIN
      log := geodb_pkg.idx_create_index(match_collect_root_id_idx);

      FOR root_id_rec IN root_id_cur LOOP
        EXECUTE 'INSERT INTO '||table_name||' (id, parent_id, root_id, geometry)
          VALUES ($1, null, $2, $3)'
          USING root_id_rec.root_id, root_id_rec.root_id,
            (geodb_pkg.match_aggregate_geometry_by_id(root_id_rec.root_id, 1));
      END LOOP;
    END;
```

Oracle

MATCH.sql (Fortführung)

```
    else
      declare
        cursor id_cur is
          select distinct id, parent_id, root_id from match_tmp_building;
      begin
        log := geodb_idx.create_index(match_collect_id_idx, false, 'nologging');

        for id_rec in id_cur loop
          execute immediate 'insert into '||table_name||' (id, parent_id, root_id,
            geometry) values (:1, :2, :3, :4)' using id_rec.id, id_rec.parent_id,
              id_rec.root_id, aggregate_geometry_by_id(id_rec.id, tolerance, 0);
        end loop;
      end;
    end if;
    -- clean up aggregate table
    execute immediate 'delete from '||table_name||' where geometry is null';
    -- create spatial index
    if match_cand_projected_spx.table_name = table_name then
      match_cand_projected_spx.srid := get_2d_srid;
      log := geodb_idx.create_index(match_cand_projected_spx, false);
    else
      match_master_projected_spx.srid := get_2d_srid;
      log := geodb_idx.create_index(match_master_projected_spx, false);
    end if;
  end;


  procedure join_cand_master(lod_cand number, lineage cityobject.lineage%type,
  lod_master number, tolerance number := 0.001)

  is
    log varchar2(4000);
  begin
    -- clean environment
    execute immediate 'truncate table match_overlap_all';
    log := geodb_idx.drop_index(match_overlap_all_spx, false);

    execute immediate 'insert all /*+ append nologging */ into match_overlap_all
       (id1, parent_id1, root_id1, area1, lod1, lineage,
        id2, parent_id2, root_id2, area2, lod2, intersection_geometry)
          select c.id id1, c.parent_id parent_id1, c.root_id root_id1,
            sdo_geom.sdo_area(c.geometry, :1) area1, :2, :3, m.id id2, m.parent_id
              parent_id2, m.root_id root_id2, sdo_geom.sdo_area(m.geometry, :4)
            area2, :5, sdo_geom.sdo_intersection(c.geometry, m.geometry, :6)
              from table(sdo_join(:7, ''geometry'', :8, ''geometry'',
              ''mask=INSIDE+CONTAINS+EQUAL+COVERS+COVEREDBY+OVERLAPBDYINTERSECT''))
                  res, '||CAND_GEOMETRY_TABLE||' c, '||MASTER_GEOMETRY_TABLE||' m
                    where c.rowid = res.rowid1 and m.rowid = res.rowid2'
      using tolerance, lod_cand, lineage, tolerance, lod_master, tolerance,
            CAND_GEOMETRY_TABLE, MASTER_GEOMETRY_TABLE;

      execute immediate 'update match_overlap_all set intersection_area =
        sdo_geom.sdo_area(intersection_geometry, :1)' using tolerance;
      execute immediate 'delete from match_overlap_all where intersection_area = 0';
      execute immediate 'update match_overlap_all set area1_cov_by_area2 =
        geodb_util.min(intersection_area / area1, 1.0), area2_cov_by_area1 =
        geodb_util.min(intersection_area / area2, 1.0)';
```

PostgreSQL / PostGIS

MATCH.sql (Fortführung)

```
  ELSE
    DECLARE
      id_cur CURSOR FOR
        SELECT DISTINCT id, parent_id, root_id FROM match_tmp_building;
    BEGIN
      log := geodb_pkg.idx_create_index(match_collect_id_idx);

      FOR id_rec IN id_cur LOOP
        EXECUTE 'INSERT INTO '||table_name||' (id, parent_id, root_id, geometry)
          VALUES ($1, $2, $3, $4)' USING id_rec.id, id_rec.parent_id,
          id_rec.root_id, (geodb_pkg.match_aggregate_geometry_by_id(id_rec.id, 0));
      END LOOP;
    END;
  END IF;
  -- clean up aggregate table
  EXECUTE 'DELETE FROM '||table_name||' WHERE geometry IS NULL';
  -- create spatial index
  IF match_cand_projected_spx.table_name = table_name THEN
    match_cand_projected_spx.srid := geodb_pkg.match_get_2d_srid();
    log := geodb_pkg.idx_create_index(match_cand_projected_spx);
  ELSE
    match_master_projected_spx.srid := geodb_pkg.match_get_2d_srid();
    log := geodb_pkg.idx_create_index(match_master_projected_spx);
  END IF;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.match_join_cand_master(
  lod_cand INTEGER, lineage cityobject.lineage%TYPE, lod_master INTEGER)
RETURNS SETOF void AS $$
DECLARE
  log VARCHAR(4000);
  match_overlap_all_spx geodb_pkg.index_obj;
BEGIN
  match_overlap_all_spx := geodb_pkg.idx_construct_spatial_2d(
   'match_overlap_all_spx', 'geodb_pkg.match_overlap_all', 'intersection_geometry');
  -- clean environment
  EXECUTE 'TRUNCATE TABLE geodb_pkg.match_overlap_all';
  log := geodb_pkg.idx_drop_index(match_overlap_all_spx);

  EXECUTE 'INSERT INTO geodb_pkg.match_overlap_all
     (id1, parent_id1, root_id1, area1, lod1, lineage,
      id2, parent_id2, root_id2, area2, lod2, intersection_geometry)
        SELECT c.id AS id1, c.parent_id AS parent_id1, c.root_id AS root_id1,
          ST_Area(c.geometry) AS area1, $1, $2, m.id AS id2, m.parent_id AS
            parent_id2, m.root_id AS root_id2, ST_Area(m.geometry) AS area2, $3,
              ST_Intersection(c.geometry, m.geometry)
         FROM geodb_pkg.MATCH_CAND_PROJECTED c, geodb_pkg.MATCH_MASTER_PROJECTED m'
           USING lod_cand, lineage, lod_master;

  EXECUTE 'UPDATE geodb_pkg.match_overlap_all SET intersection_area =
              ST_Area(intersection_geometry)';
  EXECUTE 'DELETE FROM geodb_pkg.match_overlap_all WHERE intersection_area = 0';
  EXECUTE 'UPDATE geodb_pkg.match_overlap_all SET area1_cov_by_area2 =
              geodb_pkg.util_min(intersection_area / area1, 1.0),
              area2_cov_by_area1 = geodb_pkg.util_min(intersection_area / area2,
              1.0)';
```

Oracle

MATCH.sql (Fortführung)

```
      -- create spatial index on intersection geometry
      match_overlap_all_spx.srid := get_2d_srid;
      log := geodb_idx.create_index(match_overlap_all_spx, false);
  end;


  procedure create_relevant_matches(delta_cand number, delta_master number)


  is
    log varchar2(4000);

    cursor ref_to_cand_cur is
      select id2, count(id1) as cnt_cand from match_overlap_relevant group by id2;

    cursor cand_to_ref_cur is
      select id1, count(id2) as cnt_ref from match_overlap_relevant group by id1;

  begin




  -- truncate table
    execute immediate 'truncate table match_overlap_relevant';
    log := geodb_idx.drop_index(match_result_spx, false);

    -- retrieve all match tupels with more than a user-specified percentage of area
    -- coverage
    execute immediate 'insert all /*+ append nologging */ into
      match_overlap_relevant select * from match_overlap_all where
        area1_cov_by_area2 >= :1 and area2_cov_by_area1 >= :2'
          using delta_cand, delta_master;

    -- enforce 1:1 matches between candidates and reference buildings
    for ref_to_cand_rec in ref_to_cand_cur loop
      if ref_to_cand_rec.cnt_cand > 1 then
        execute immediate 'delete from match_overlap_relevant where id2=:1'
                          using ref_to_cand_rec.id2;
      end if;
    end loop;

    for cand_to_ref_rec in cand_to_ref_cur loop
      if cand_to_ref_rec.cnt_ref > 1 then
        execute immediate 'delete from match_overlap_relevant where id1=:1'
                          using cand_to_ref_rec.id1;
      end if;
    end loop;

    -- create spatial index on intersection geometry
    match_result_spx.srid := get_2d_srid;
    log := geodb_idx.create_index(match_result_spx, false);
  exception
      when others then
        dbms_output.put_line('create_relevant_matches: ' || SQLERRM);
  end;
```

PostgreSQL / PostGIS

MATCH.sql (Fortführung)

```
  -- create spatial index on intersection geometry
  match_overlap_all_spx.srid := geodb_pkg.match_get_2d_srid();
  log := geodb_pkg.idx_create_index(match_overlap_all_spx);
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.match_create_relevant_matches(delta_cand DOUBLE
PRECISION, delta_master DOUBLE PRECISION)
RETURNS SETOF void AS $$
DECLARE
  log VARCHAR(4000);
  match_result_spx geodb_pkg.index_obj;
  ref_to_cand_cur CURSOR FOR
    SELECT id2, count(id1) AS cnt_cand FROM geodb_pkg.match_overlap_relevant
      GROUP BY id2;
  cand_to_ref_cur CURSOR FOR
    SELECT id1, count(id2) AS cnt_ref FROM geodb_pkg.match_overlap_relevant
      GROUP BY id1;
BEGIN
  match_result_spx := geodb_pkg.idx_construct_spatial_2d(
    'match_overlap_relevant_spx', 'geodb_pkg.match_overlap_relevant',
    'intersection_geometry');

  -- TRUNCATE TABLE
  EXECUTE 'TRUNCATE TABLE geodb_pkg.match_overlap_relevant';
  log := geodb_pkg.idx_drop_index(match_result_spx);

  -- retrieve all match tupels with more than a user-specified percentage of area
  -- coverage
  EXECUTE 'INSERT INTO geodb_pkg.match_overlap_relevant
    SELECT * FROM geodb_pkg.match_overlap_all
      WHERE area1_cov_by_area2 >= $1 AND area2_cov_by_area1 >= $2'
        USING delta_cand, delta_master;

  -- enforce 1:1 matches between candidates and reference buildings
  FOR ref_to_cand_rec IN ref_to_cand_cur LOOP
    IF ref_to_cand_rec.cnt_cand > 1 THEN
      EXECUTE 'DELETE FROM geodb_pkg.match_overlap_relevant WHERE id2=$1'
                 USING ref_to_cand_rec.id2;
    END IF;
  END LOOP;

  FOR cand_to_ref_rec IN cand_to_ref_cur LOOP
    IF cand_to_ref_rec.cnt_ref > 1 THEN
      EXECUTE 'DELETE FROM geodb_pkg.match_overlap_relevant WHERE id1=$1'
                 USING cand_to_ref_rec.id1;
    END IF;
  END LOOP;

  -- create spatial index on intersection geometry
  match_result_spx.srid := geodb_pkg.match_get_2d_srid();
  log := geodb_pkg.idx_create_index(match_result_spx);
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'create_relevant_matches: %', SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

MATCH.sql (Fortführung)

```
procedure clear_matching_tables
is
begin
  execute immediate 'truncate table match_overlap_all';
  execute immediate 'truncate table match_overlap_relevant';
  execute immediate 'truncate table match_master_projected';
  execute immediate 'truncate table match_cand_projected';
  execute immediate 'truncate table match_collect_geom';
  execute immediate 'truncate table match_tmp_building';
  execute immediate 'truncate table merge_collect_geom';
  execute immediate 'truncate table merge_container_ids';
exception
  when others then
    dbms_output.put_line('clean_matching_tables: ' || SQLERRM);
end;


function aggregate_mbr(table_name varchar2)
return mdsys.sdo_geometry
is
  aggr_mbr mdsys.sdo_geometry;
  srid number;
begin
  execute immediate 'select srid from database_srs' into srid;
  execute immediate 'select sdo_aggr_mbr(geometry) from '||table_name||'' into
    aggr_mbr;
  aggr_mbr.sdo_srid := srid;
  return aggr_mbr;
exception
  when others then
    return null;
end;


function aggregate_geometry_by_id(id number, tolerance number := 0.001,
aggregate_building number := 1)
return mdsys.sdo_geometry
is
  aggr_geom mdsys.sdo_geometry;
  attr string(10);
begin
  if aggregate_building > 0 then
    attr := 'root_id';
  else
    attr := 'id';
  end if;
  execute immediate 'select sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom, :1))
    from (select sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom, :2)) aggr_geom
      from (select sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom, :3)) aggr_geom
        from (select sdo_aggr_union(mdsys.sdoaggrtype(geometry, :4)) aggr_geom
          from match_collect_geom where '||attr||'=:5 group by mod(rownum, 1000))
             group by mod (rownum, 100)) group by mod (rownum, 10))'
    into aggr_geom using tolerance, tolerance, tolerance, tolerance, id;
  return aggr_geom;
exception
  when others then
    return null;
end;
```

PostgreSQL / PostGIS

MATCH.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.match_clear_matching_tables()
RETURNS SETOF void AS $$
BEGIN
  EXECUTE 'TRUNCATE TABLE geodb_pkg.match_overlap_all';
  EXECUTE 'TRUNCATE TABLE geodb_pkg.match_overlap_relevant';
  EXECUTE 'TRUNCATE TABLE geodb_pkg.match_master_projected';
  EXECUTE 'TRUNCATE TABLE geodb_pkg.match_cand_projected';
  EXECUTE 'TRUNCATE TABLE geodb_pkg.match_collect_geom';
  EXECUTE 'TRUNCATE TABLE match_tmp_building';
  EXECUTE 'TRUNCATE TABLE geodb_pkg.merge_collect_geom';
  EXECUTE 'TRUNCATE TABLE geodb_pkg.merge_container_ids';
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'clean_matching_tables: %', SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.match_aggregate_mbr(table_name VARCHAR)
RETURNS GEOMETRY AS $$
DECLARE
  aggr_mbr GEOMETRY;
  srid INTEGER;
BEGIN
  EXECUTE 'SELECT srid FROM database_srs' INTO srid;
  EXECUTE 'SELECT ST_Union(geometry) FROM '||table_name||'' INTO aggr_mbr;
  PERFORM ST_SetSRID(aggr_mbr, srid);

  RETURN aggr_mbr;
  EXCEPTION
    WHEN OTHERS THEN
      RETURN null;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.match_aggregate_geometry_by_id(id INTEGER,
aggregate_building INTEGER DEFAULT 1)
RETURNS GEOMETRY AS $$
DECLARE
  aggr_geom GEOMETRY;
  attr VARCHAR(10);
BEGIN
  IF aggregate_building > 0 THEN
    attr := 'root_id';
  ELSE
    attr := 'id';
  END IF;
  EXECUTE 'SELECT ST_Union(geometry) FROM geodb_pkg.match_collect_geom
           WHERE '||attr||'=$1' INTO aggr_geom USING id;
  -- In Oracle kann die Union-Funktion so strukturiert werden, dass alle Geometrien
  -- zunächst in kleineren Gruppen zusammengefasst werden, welche dann in größere
  -- Gruppen aufgehen usw. Das ist deutlich performanter als das Verschmelzen aller
  -- Geometrien in einem einzigen Schritt, wie es PostGIS bisher tut.
  RETURN aggr_geom;
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE '%: %', id, SQLERRM;
      RETURN null;
END; $$ LANGUAGE plpgsql;
```

Oracle

MATCH.sql (Fortführung)

```
  function get_2d_srid return number

  is
    srid number;
  begin
    if geodb_util.is_db_coord_ref_sys_3d = 1 then
      srid := null;
    else
      execute immediate 'select srid from database_srs' into srid;
    end if;

    return srid;
  end;

END geodb_match;
/
```

MERGE.sql

```
set term off;
set serveroutput off;

drop table merge_collect_geom;
create global temporary table merge_collect_geom
   ( building_id number,
     geometry_id number,
     cityobject_id number
   ) on commit preserve rows;

drop table merge_container_ids;
create table merge_container_ids
   ( building_id number,
     container_id number
   ) nologging;

set term on;
set serveroutput on;

CREATE OR REPLACE PACKAGE geodb_merge
AS
  procedure process_matches(lod_src number, lod_dst number, name_mode number,
    delimiter varchar2);
  . . .
END geodb_merge;
/

CREATE OR REPLACE PACKAGE BODY geodb_merge AS
  -- declaration of private indexes
  merge_geom_building_id_idx index_obj := index_obj.construct_normal(
    'merge_geom_building_id_idx', 'merge_collect_geom', 'building_id');
  merge_geom_geometry_id_idx index_obj := index_obj.construct_normal(
    'merge_geom_geometry_id_idx', 'merge_collect_geom', 'geometry_id');
  merge_cont_building_id_idx index_obj := index_obj.construct_normal(
    'merge_cont_building_id_idx', 'merge_container_ids', 'building_id');
  merge_cont_id_idx index_obj := index_obj.construct_normal(
    'merge_cont_id_idx', 'merge_container_ids', 'container_id');
```

PostgreSQL / PostGIS

MATCH.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.match_get_2d_srid()
RETURNS INTEGER AS $$
DECLARE
  srid INTEGER;
BEGIN
  IF geodb_pkg.util_is_db_coord_ref_sys_3d() = 1 THEN
    srid := null;
  ELSE
    EXECUTE 'SELECT srid FROM database_srs' INTO srid;
  END IF;

  RETURN srid;
END; $$ LANGUAGE plpgsql;
```

MERGE.sql

```
-- Die Temporäre Tabelle wird in jener Funktion definiert, die als erste vom
-- Importer/Exporter aufgerufen wird (siehe MATCH.sql).
```

```
DROP TABLE IF EXISTS geodb_pkg.merge_container_ids;
CREATE TABLE geodb_pkg.merge_container_ids (
  building_id INTEGER,
  container_id INTEGER
);
```

Oracle

MERGE.sql (Fortführung)

```
procedure process_matches(lod_src number, lod_dst number, name_mode number,
delimiter varchar2)
is
begin
  -- find relevant matches
  collect_all_geometry(lod_src);
  remove_geometry_from_cand(lod_src);
  create_and_put_container(lod_dst, name_mode, delimiter);
  move_appearance();
  move_geometry();
  delete_head_of_merge_geometry();
  commit;
exception
  when others then
    dbms_output.put_line('process_matches: ' || SQLERRM);
end;


procedure collect_all_geometry(lod number)

is
  log varchar2(4000);


begin
```

```
  execute immediate 'truncate table merge_collect_geom';
  log := geodb_idx.drop_index(merge_geom_building_id_idx, false);
  log := geodb_idx.drop_index(merge_geom_geometry_id_idx, false);

  -- retrieve all building and building part geometry
  execute immediate 'insert all /*+ append nologging */ into merge_collect_geom
    select b.building_root_id, b.lod'||to_char(lod)||'_geometry_id, b.id
      from building b, match_overlap_relevant m
        where b.building_root_id = m.id1
          and b.lod'||to_char(lod)||'_geometry_id is not null';

  if lod >= 2 then
    -- retrieve relevant building installation geometry
    execute immediate 'insert all /*+ append nologging */ into merge_collect_geom
      select b.building_root_id, i.lod'||to_char(lod)||'_geometry_id, i.id
        from match_overlap_relevant m, building_installation i, building b
          where i.building_id = b.id
          and b.building_root_id = m.id1
          and i.is_external = 1
          and i.lod'||to_char(lod)||'_geometry_id is not null';
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.merge_process_matches(
   lod_src NUMERIC, lod_dst NUMERIC, name_mode NUMERIC, delimiter VARCHAR)
RETURNS SETOF void AS $$
BEGIN
  -- find relevant matches
  PERFORM geodb_pkg.merge_collect_all_geometry(lod_src);
  PERFORM geodb_pkg.merge_remove_geometry_from_cand(lod_src);
  PERFORM geodb_pkg.merge_create_and_put_container(lod_dst, name_mode, delimiter);
  PERFORM geodb_pkg.merge_move_appearance();
  PERFORM geodb_pkg.merge_move_geometry();
  PERFORM geodb_pkg.merge_delete_head_of_merge_geometry();
  COMMIT;
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'process_matches: %', SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.merge_collect_all_geometry(lod NUMERIC)
RETURNS SETOF void AS $$
DECLARE
  log VARCHAR(4000);
  merge_geom_building_id_idx geodb_pkg.index_obj;
  merge_geom_geometry_id_idx geodb_pkg.index_obj;
BEGIN
  -- creates the temporary table merge_collect_geom
  EXECUTE 'CREATE GLOBAL TEMPORARY TABLE merge_collect_geom(
    building_id INTEGER,
    geometry_id INTEGER,
    cityobject_id INTEGER
    ) ON COMMIT PRESERVE ROWS';

  merge_geom_building_id_idx := geodb_pkg.idx_construct_normal(
    'merge_geom_building_id_idx', 'merge_collect_geom', 'building_id');
  merge_geom_geometry_id_idx := geodb_pkg.idx_construct_normal(
    'merge_geom_geometry_id_idx', 'merge_collect_geom', 'geometry_id');

  EXECUTE 'TRUNCATE TABLE merge_collect_geom';
  log := geodb_pkg.idx_drop_index(merge_geom_building_id_idx);
  log := geodb_pkg.idx_drop_index(merge_geom_geometry_id_idx);

-- retrieve all building and building part geometry
EXECUTE 'INSERT INTO merge_collect_geom
  SELECT b.building_root_id, b.lod'||lod||'_geometry_id, b.id
    FROM building b, geodb_pkg.match_overlap_relevant m
      WHERE b.building_root_id = m.id1
        AND b.lod'||lod||'_geometry_id IS NOT NULL';

IF lod >= 2 THEN
  -- retrieve relevant building installation geometry
  EXECUTE 'INSERT INTO merge_collect_geom
    SELECT b.building_root_id, i.lod'||lod||'_geometry_id, i.id
      FROM geodb_pkg.match_overlap_relevant m, building_installation i, building b
        WHERE i.building_id = b.id
          AND b.building_root_id = m.id1
          AND i.is_external = 1
          AND i.lod'||lod||'_geometry_id IS NOT NULL';
```

Oracle

MERGE.sql (Fortführung)

```
   -- retrieve surfaces from relevant thematic surfaces
      execute immediate 'insert all /*+ append nologging */ into merge_collect_geom
        select  b.building_root_id, t.lod'||to_char(lod)||'_multi_surface_id, t.id
          from match_overlap_relevant m, thematic_surface t, building b
            where t.building_id = b.id and b.building_root_id = m.id1
              and t.lod'||to_char(lod)||'_multi_surface_id is not null';
    end if;

    if lod >= 3 then
      -- retrieve all openings of all thematic surfaces beloning to all buildings
      -- and building parts
      execute immediate 'insert all /*+ append nologging */ into merge_collect_geom
        select b.building_root_id, o.lod'||to_char(lod)||'_multi_surface_id, o.id
          from match_overlap_relevant m, thematic_surface t, building b, opening o,
          opening_to_them_surface ot
            where t.building_id = b.id and b.building_root_id = m.id1
              and ot.thematic_surface_id = t.id and o.id = ot.opening_id
              and o.lod'||to_char(lod)||'_multi_surface_id is not null';
    end if;

    if lod >= 4 then
      -- room
      execute immediate 'insert all /*+ append nologging */ into merge_collect_geom
        select b.building_root_id, r.lod4_geometry_id, r.id
          from match_overlap_relevant m, room r, building b
          where r.building_id = b.id
                and b.building_root_id = m.id1
                and r.lod4_geometry_id is not null';

      -- building furniture (in rooms) --bei lod r in f geändert
      execute immediate 'insert all /*+ append nologging */ into merge_collect_geom
        select b.building_root_id, f.lod4_geometry_id, f.id
          from match_overlap_relevant m, room r, building b, building_furniture f
          where r.building_id = b.id
                and b.building_root_id = m.id1
                and f.room_id = r.id
                and f.lod4_geometry_id is not null';

      -- retrieve relevant internal (or external) building installation geometry (in
      -- rooms)
      execute immediate 'insert all /*+ append nologging */ into merge_collect_geom
        select b.building_root_id, i.lod4_geometry_id, i.id
          from match_overlap_relevant m, building_installation i, building b, room r
          where r.building_id = b.id
                and i.room_id = r.id
                and b.building_root_id = m.id1
                and i.lod4_geometry_id is not null';

      -- retrieve surfaces from relevant thematic surfaces (in rooms)
      execute immediate 'insert all /*+ append nologging */ into merge_collect_geom
        select  b.building_root_id, t.lod4_multi_surface_id, t.id
          from match_overlap_relevant m, thematic_surface t, building b, room r
          where r.building_id = b.id
                and t.room_id = r.id
                and b.building_root_id = m.id1
                and t.lod4_multi_surface_id is not null';
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
-- retrieve surfaces from relevant thematic surfaces
EXECUTE 'INSERT INTO merge_collect_geom
  SELECT  b.building_root_id, t.lod'||lod||'_multi_surface_id, t.id
    FROM geodb_pkg.match_overlap_relevant m, thematic_surface t, building b
      WHERE t.building_id = b.id AND b.building_root_id = m.id1
        AND t.lod'||lod||'_multi_surface_id IS NOT NULL';
END IF;

    IF lod >= 3 THEN
      -- retrieve all openings of all thematic surfaces beloning to all buildings and
      -- building parts
      EXECUTE 'INSERT INTO merge_collect_geom
        SELECT b.building_root_id, o.lod'||lod||'_multi_surface_id, o.id
          FROM geodb_pkg.match_overlap_relevant m, thematic_surface t, building b,
          opening o, opening_to_them_surface ot
            WHERE t.building_id = b.id AND b.building_root_id = m.id1
              AND ot.thematic_surface_id = t.id AND o.id = ot.opening_id
              AND o.lod'||lod||'_multi_surface_id IS NOT NULL';
    END IF;

    IF lod >= 4 THEN
      -- room
      EXECUTE 'INSERT INTO merge_collect_geom
        SELECT b.building_root_id, r.lod4_geometry_id, r.id
          FROM geodb_pkg.match_overlap_relevant m, room r, building b
            WHERE r.building_id = b.id
              AND b.building_root_id = m.id1
              AND r.lod4_geometry_id IS NOT NULL';

      -- building furniture (in rooms) --if lod r is changed to f
      EXECUTE 'INSERT INTO merge_collect_geom
        SELECT b.building_root_id, f.lod4_geometry_id, f.id
          FROM geodb_pkg.match_overlap_relevant m, room r, building b,
          building_furniture f
            WHERE r.building_id = b.id AND b.building_root_id = m.id1
              AND f.room_id = r.id
              AND f.lod4_geometry_id IS NOT NULL';

      -- retrieve relevant internal (or external) building installation geometry (in
      -- rooms)
      EXECUTE 'INSERT INTO merge_collect_geom
        SELECT b.building_root_id, i.lod4_geometry_id, i.id
          FROM geodb_pkg.match_overlap_relevant m, building_installation i, building
          b, room r
            WHERE r.building_id = b.id AND i.room_id = r.id
              AND b.building_root_id = m.id1
              AND i.lod4_geometry_id IS NOT NULL';

      -- retrieve surfaces from relevant thematic surfaces (in rooms)
      EXECUTE 'INSERT INTO merge_collect_geom
        SELECT  b.building_root_id, t.lod4_multi_surface_id, t.id
          FROM geodb_pkg.match_overlap_relevant m, thematic_surface t, building b,
          room r
            WHERE r.building_id = b.id AND t.room_id = r.id
              AND b.building_root_id = m.id1
              AND t.lod4_multi_surface_id IS NOT NULL';
```

Oracle

MERGE.sql (Fortführung)

```
      -- retrieve all openings of all thematic surfaces beloning to all rooms in all
      -- buildings and building parts
       execute immediate 'insert all /*+ append nologging */ into merge_collect_geom
         select b.building_root_id, o.lod4_multi_surface_id, o.id
           from match_overlap_relevant m, thematic_surface t, building b, opening o,
           opening_to_them_surface ot, room r
             where r.building_id = b.id and t.room_id = r.id
               and b.building_root_id = m.id1 and ot.thematic_surface_id = t.id
               and o.id = ot.opening_id and o.lod4_multi_surface_id is not null';

       -- retrieve relevant internal building installation geometry
       execute immediate 'insert all /*+ append nologging */ into merge_collect_geom
         select b.building_root_id, i.lod4_geometry_id, i.id
           from match_overlap_relevant m, building_installation i, building b
           where i.building_id = b.id and b.building_root_id = m.id1
             and i.is_external = 0 and i.lod4_geometry_id is not null';
     end if;

     log := geodb_idx.create_index(merge_geom_building_id_idx, false, 'nologging');
     log := geodb_idx.create_index(merge_geom_geometry_id_idx, false, 'nologging');
   exception
       when others then
         dbms_output.put_line('collect_all_geometry: ' || SQLERRM);
   end;


   procedure remove_geometry_from_cand(lod number)
   is
   begin
     -- retrieve all building and building part geometry
     execute immediate 'update building b
       set b.lod'||to_char(lod)||'_geometry_id = null
       where b.building_root_id in (select id1 from match_overlap_relevant)';

     if lod >= 2 then
       -- retrieve relevant building installation geometry
       execute immediate 'update building_installation i
         set i.lod'||to_char(lod)||'_geometry_id = null
         where i.building_id in (select b.id from building b, match_overlap_relevant
           m where b.building_root_id = m.id1) and i.is_external = 1';

       -- retrieve surfaces from relevant thematic surfaces
       execute immediate 'update thematic_surface t
       set t.lod'||to_char(lod)||'_multi_surface_id = null
       where t.building_id in (select b.id from building b, match_overlap_relevant m
         where b.building_root_id = m.id1)';
     end if;

     if lod >= 3 then
       -- retrieve all openings of all thematic surfaces beloning to all buildings
       -- and building parts
       execute immediate 'update opening o
         set o.lod'||to_char(lod)||'_multi_surface_id = null
         where o.id in (select ot.opening_id from match_overlap_relevant m,
           thematic_surface t, building b, opening_to_them_surface ot
           where ot.thematic_surface_id = t.id and t.building_id = b.id
           and b.building_root_id = m.id1)';
     end if;
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
    -- retrieve all openings of all thematic surfaces beloning to all rooms in all
    -- buildings and building parts
    EXECUTE 'INSERT INTO merge_collect_geom
      SELECT b.building_root_id, o.lod4_multi_surface_id, o.id
        FROM geodb_pkg.match_overlap_relevant m, thematic_surface t, building b,
        opening o, opening_to_them_surface ot, room r
          WHERE r.building_id = b.id AND t.room_id = r.id
            AND b.building_root_id = m.id1 AND ot.thematic_surface_id = t.id
            AND o.id = ot.opening_id AND o.lod4_multi_surface_id IS NOT NULL';

    -- retrieve relevant internal building installation geometry
    EXECUTE 'INSERT INTO merge_collect_geom
      SELECT b.building_root_id, i.lod4_geometry_id, i.id
        FROM geodb_pkg.match_overlap_relevant m, building_installation i, building b
          WHERE i.building_id = b.id AND b.building_root_id = m.id1
            AND i.is_external = 0 AND i.lod4_geometry_id IS NOT NULL';
  END IF;

  log := geodb_pkg.idx_create_index(merge_geom_building_id_idx);
  log := geodb_pkg.idx_create_index(merge_geom_geometry_id_idx);
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'collect_all_geometry: %', SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.merge_remove_geometry_from_cand(lod NUMERIC)
RETURNS SETOF void AS $$
BEGIN
  -- retrieve all building and building part geometry
  EXECUTE 'UPDATE building SET lod'||lod||'_geometry_id = null
    WHERE building_root_id IN
     (SELECT id1 FROM geodb_pkg.match_overlap_relevant)';

  IF lod >= 2 THEN
    -- retrieve relevant building installation geometry
    EXECUTE 'UPDATE building_installation SET lod'||lod||'_geometry_id = null
      WHERE building_id IN (SELECT b.id FROM building b,
        geodb_pkg.match_overlap_relevant m WHERE b.building_root_id = m.id1)
        AND i.is_external = 1';

    -- retrieve surfaces from relevant thematic surfaces
    EXECUTE 'UPDATE thematic_surface SET lod'||lod||'_multi_surface_id = null
      WHERE building_id IN
        (SELECT b.id FROM building b, geodb_pkg.match_overlap_relevant m
          WHERE b.building_root_id = m.id1)';
  END IF;

  IF lod >= 3 THEN
    -- retrieve all openings of all thematic surfaces beloning to all buildings and
    -- building parts
    EXECUTE 'UPDATE opening
      SET lod'||lod||'_multi_surface_id = null
      WHERE id IN (SELECT ot.opening_id FROM geodb_pkg.match_overlap_relevant m,
        thematic_surface t, building b, opening_to_them_surface ot
          WHERE ot.thematic_surface_id = t.id AND t.building_id = b.id
            AND b.building_root_id = m.id1)';
  END IF;
```

Oracle

MERGE.sql (Fortführung)

```
  if lod >= 4 then
     -- room
     execute immediate 'update room r
     set r.lod4_geometry_id = null
     where r.building_id in
       (select b.id from match_overlap_relevant m, building b
          where b.building_root_id = m.id1)';

     -- building furniture (in rooms) --bei lod r in f geändert
     execute immediate 'update building_furniture f
     set f.lod4_geometry_id = null
     where f.room_id in
       (select r.id from match_overlap_relevant m, room r, building b
          where r.building_id = b.id
            and b.building_root_id = m.id1)';

     -- retrieve relevant internal (or external) building installation geometry
     -- (in rooms)
     execute immediate 'update building_installation i
     set i.lod4_geometry_id = null
     where i. room_id in
       (select r.id from match_overlap_relevant m, building b, room r
          where r.building_id = b.id
            and b.building_root_id = m.id1)';

     -- retrieve surfaces from relevant thematic surfaces (in rooms)
     execute immediate 'update thematic_surface t
     set t.lod4_multi_surface_id = null
     where t.room_id in
       (select r.id from match_overlap_relevant m, building b, room r
          where r.building_id = b.id
            and b.building_root_id = m.id1)';

     -- retrieve all openings of all thematic surfaces beloning to all rooms in
     -- all buildings and building parts
     execute immediate 'update opening o
     set o.lod4_multi_surface_id = null
     where o.id in
       (select ot.opening_id from match_overlap_relevant m, thematic_surface t,
          building b, opening_to_them_surface ot, room r
            where r.building_id = b.id
              and t.room_id = r.id
              and b.building_root_id = m.id1
              and ot.thematic_surface_id = t.id)';

     -- retrieve relevant internal building installation geometry
     execute immediate 'update building_installation i
     set i.lod4_geometry_id = null
     where i.is_external = 0
       and i.building_id in
         (select b.id from match_overlap_relevant m, building b
            where b.building_root_id = m.id1)';
     end if;
 exception
     when others then
       dbms_output.put_line('remove_geometry_from_cand: ' || SQLERRM);
 end;
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
IF lod >= 4 THEN
    -- room
    EXECUTE 'UPDATE room SET lod4_geometry_id = null
      WHERE building_id IN
        (SELECT b.id FROM geodb_pkg.match_overlap_relevant m, building b
           WHERE b.building_root_id = m.id1)';


    -- building furniture (in rooms) --if lod r is changed to f
    EXECUTE 'UPDATE building_furniture SET lod4_geometry_id = null
      WHERE room_id IN
        (SELECT r.id FROM geodb_pkg.match_overlap_relevant m, room r, building b
           WHERE r.building_id = b.id
             AND b.building_root_id = m.id1)';


    -- retrieve relevant internal (or external) building installation geometry (in
    -- rooms)
    EXECUTE 'UPDATE building_installation SET lod4_geometry_id = null
      WHERE room_id IN
        (SELECT r.id FROM geodb_pkg.match_overlap_relevant m, building b, room r
           WHERE r.building_id = b.id
             AND b.building_root_id = m.id1)';


    -- retrieve surfaces from relevant thematic surfaces (in rooms)
    EXECUTE 'UPDATE thematic_surface SET lod4_multi_surface_id = null
      WHERE room_id IN
        (SELECT r.id FROM geodb_pkg.match_overlap_relevant m, building b, room r
           WHERE r.building_id = b.id
             AND b.building_root_id = m.id1)';


    -- retrieve all openings of all thematic surfaces beloning to all rooms in all
    -- buildings and building parts
    EXECUTE 'UPDATE opening SET lod4_multi_surface_id = null
      WHERE id IN
        (SELECT ot.opening_id FROM geodb_pkg.match_overlap_relevant m,
         thematic_surface t, building b, opening_to_them_surface ot, room r
           WHERE r.building_id = b.id
             AND t.room_id = r.id
             AND b.building_root_id = m.id1
             AND ot.thematic_surface_id = t.id)';


    -- retrieve relevant internal building installation geometry
    EXECUTE 'UPDATE building_installation SET lod4_geometry_id = null
      WHERE is_external = 0 AND building_id IN
        (SELECT b.id FROM geodb_pkg.match_overlap_relevant m, building b
           WHERE b.building_root_id = m.id1)';


  END IF;
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'remove_geometry_from_cand: %', SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

MERGE.sql (Fortführung)

```
procedure create_and_put_container(lod number, name_mode number, delimiter
varchar2)

  is
    seq_val number;
    old_geometry number;
    log varchar2(4000);
    cursor building_id_cur is
      select id1 from match_overlap_relevant;


  begin





  execute immediate 'truncate table merge_container_ids';
    log := geodb_idx.drop_index(merge_cont_building_id_idx, false);
    log := geodb_idx.drop_index(merge_cont_id_idx, false);

    -- iterate through all affected buildings
    for building_id_rec in building_id_cur loop
      -- create geometry id and register in container
      execute immediate 'select surface_geometry_seq.nextval from dual'
        into seq_val;
      execute immediate 'insert into merge_container_ids (building_id,
        container_id) values (:1, :2)' using building_id_rec.id1, seq_val;

      -- retrieve and delete old geometry
      begin
        execute immediate 'select b.lod'||to_char(lod)||'_geometry_id from building
          b where b.id = (select id2 from match_overlap_relevant where id1 = :1)'
          into old_geometry using building_id_rec.id1;
      exception
        when others then
          old_geometry := 0;
      end;

      -- create new multisurface as root element of new merge geometry
      execute immediate 'insert into surface_geometry (id, parent_id, root_id,
        is_solid, is_composite, is_triangulated, is_xlink, is_reverse, geometry)
        values (:1, null, :2, 0, 0, 0, 0, 0, null)' using seq_val, seq_val;

      -- set building geometry to new multisurface and process name
      if name_mode=1 then
        -- ignore cand name
        execute immediate 'update building b
          set b.lod'||to_char(lod)||'_geometry_id = :1
          where b.id = (select id2 from match_overlap_relevant where id1 = :2)'
            using seq_val, building_id_rec.id1;
      elsif name_mode=2 then
        -- replace master name with cand name
        execute immediate 'update building b
          set b.lod'||to_char(lod)||'_geometry_id = :1,
          b.name = (select name from building where id = :2)
          where b.id = (select id2 from match_overlap_relevant where id1 = :3)'
            using seq_val, building_id_rec.id1, building_id_rec.id1;
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.merge_create_and_put_container(
  lod NUMERIC, name_mode NUMERIC, delimiter VARCHAR)
RETURNS SETOF void AS $$
DECLARE
  seq_val INTEGER;
  old_geometry NUMERIC;
  log VARCHAR(4000);
  merge_cont_building_id_idx geodb_pkg.index_obj;
  merge_cont_id_idx geodb_pkg.index_obj;
  building_id_cur CURSOR FOR
    SELECT id1 FROM geodb_pkg.match_overlap_relevant;
BEGIN
  merge_cont_building_id_idx := geodb_pkg.idx_construct_normal(
    'merge_cont_building_id_idx', 'geodb_pkg.merge_container_ids', 'building_id');
  merge_cont_id_idx := geodb_pkg.idx_construct_normal(
    'merge_cont_id_idx', 'geodb_pkg.merge_container_ids', 'container_id');

  EXECUTE 'TRUNCATE TABLE geodb_pkg.merge_container_ids';
  log := geodb_pkg.idx_drop_index(merge_cont_building_id_idx);
  log := geodb_pkg.idx_drop_index(merge_cont_id_idx);

  -- iterate through all affected buildings
  FOR building_id_rec IN building_id_cur LOOP
    -- create geometry id and register in container
    EXECUTE 'SELECT nextval(''surface_geometry_id_seq'')' INTO seq_val;

    EXECUTE 'INSERT INTO geodb_pkg.merge_container_ids (building_id, container_id)
      VALUES ($1, $2)' USING building_id_rec.id1, seq_val;

    -- retrieve and delete old geometry
    BEGIN
      EXECUTE 'SELECT b.lod'||lod||'_geometry_id FROM building b
        WHERE b.id = (SELECT id2 FROM geodb_pkg.match_overlap_relevant WHERE id1 =
          $1)' INTO old_geometry USING building_id_rec.id1;
      EXCEPTION
        WHEN OTHERS THEN
          old_geometry := 0;
    END;

    -- create new multisurface as root element of new merge geometry
    EXECUTE 'INSERT INTO surface_geometry (id, parent_id, root_id, is_solid,
      is_composite, is_triangulated, is_xlink, is_reverse, geometry)
      VALUES ($1, null, $2, 0, 0, 0, 0, 0, null)' USING seq_val, seq_val;

    -- set building geometry to new multisurface and process name
    IF name_mode=1 THEN
      -- ignore cand name
      EXECUTE 'UPDATE building SET lod'||lod||'_geometry_id = $1
        WHERE id = (SELECT id2 FROM geodb_pkg.match_overlap_relevant WHERE id1 =
          $2)' USING seq_val, building_id_rec.id1;

    ELSIF name_mode=2 THEN
      -- replace master name with cand name
      EXECUTE 'UPDATE building SET lod'||lod||'_geometry_id = $1,
        name = (SELECT name FROM building WHERE id = $2)
          WHERE id = (SELECT id2 FROM geodb_pkg.match_overlap_relevant WHERE id1 =
            $3)' USING seq_val, building_id_rec.id1, building_id_rec.id1;
```

Oracle

MERGE.sql (Fortführung)

```
      else
        -- append cand name to master
        execute immediate 'update building b
          set b.lod'||to_char(lod)||'_geometry_id = :1,
            b.name = concat(b.name, nullif(concat(:2, (select name from building
              where id = :3)), :4)) where b.id = (select id2 from
                match_overlap_relevant where id1 = :4)'
                  using seq_val, delimiter, building_id_rec.id1, delimiter,
                          building_id_rec.id1;
      end if;

      -- delete old geometry
      if old_geometry > 0 then
        geodb_delete.delete_surface_geometry(old_geometry);
      end if;
    end loop;

    log := geodb_idx.create_index(merge_cont_building_id_idx, false, 'nologging');
    log := geodb_idx.create_index(merge_cont_id_idx, false, 'nologging');
  exception
    when others then
      dbms_output.put_line('create_and_put_container: ' || SQLERRM);
  end;


  procedure move_appearance

  is
    geom_hierachies number;
    app_id number;
    seq_val number;
    building_id number;
    not_version_enabled boolean;
    cursor building_cur is
      select building_id, count(geometry_id) as cnt_hierarchies from
        merge_collect_geom group by building_id;
  begin
    not_version_enabled := geodb_util.versioning_table('APPEAR_TO_SURFACE_DATA') <>
                              'ON';
    -- iterate through all building matches
    for building_rec in building_cur loop
      declare
        cursor app_cur is
          select distinct a.id, a.theme, a.description, sd.id as sd_id
            from merge_collect_geom cg, surface_data sd, appear_to_surface_data
              asd, appearance a where a.cityobject_id=building_rec.building_id
              and asd.appearance_id=a.id and sd.id=asd.surface_data_id
              and (select count(*) from textureparam t where
                t.surface_data_id=sd.id) > 0 order by a.id;

        cursor geom_cur is
          select distinct tp.surface_geometry_id as geometry_id, tp.surface_data_id
            as sd_id, cg.geometry_id as hierarchy_id
            from merge_collect_geom cg, textureparam tp
            where cg.building_id=building_rec.building_id
            and tp.surface_geometry_id=cg.geometry_id;
      begin
        app_id := -1;
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
    ELSE
      -- append cand name to master
      EXECUTE 'UPDATE building SET lod'||lod||'_geometry_id = $1,
        name = concat(name, nullif(concat($2, (SELECT name FROM building WHERE id =
          $3)), $4)) WHERE id = (SELECT id2 FROM geodb_pkg.match_overlap_relevant
             WHERE id1 = $5)' USING seq_val, delimiter, building_id_rec.id1,
                                  delimiter, building_id_rec.id1;

    END IF;

    -- delete old geometry
    IF old_geometry > 0 THEN
      PERFORM geodb_pkg.del_delete_surface_geometry(old_geometry);
    END IF;
  END LOOP;

  log := geodb_pkg.idx_create_index(merge_cont_building_id_idx);
  log := geodb_pkg.idx_create_index(merge_cont_id_idx);

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'create_and_put_container: %', SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.merge_move_appearance()
RETURNS SETOF void AS $$
DECLARE
  geom_hierachies NUMERIC;
  app_id INTEGER;
  seq_val NUMERIC;
  building_id INTEGER;
  building_cur CURSOR FOR
    SELECT mcg.building_id, count(mcg.geometry_id) AS cnt_hierarchies FROM
      merge_collect_geom mcg GROUP BY mcg.building_id;

BEGIN


  -- iterate through all building matches
  FOR building_rec IN building_cur LOOP
    DECLARE
      app_cur CURSOR FOR
        SELECT DISTINCT a.id, a.theme, a.description, sd.id AS sd_id
          FROM merge_collect_geom cg, surface_data sd, appear_to_surface_data asd,
            appearance a WHERE a.cityobject_id=building_rec.building_id
              AND asd.appearance_id=a.id AND sd.id=asd.surface_data_id
              AND (SELECT count(*) FROM textureparam t WHERE
                t.surface_data_id=sd.id) > 0 ORDER BY a.id;

      geom_cur CURSOR FOR
        SELECT DISTINCT tp.surface_geometry_id AS geometry_id, tp.surface_data_id AS
          sd_id, cg.geometry_id AS hierarchy_id
          FROM merge_collect_geom cg, textureparam tp
            WHERE cg.building_id=building_rec.building_id
              AND tp.surface_geometry_id=cg.geometry_id;
    BEGIN
      app_id := -1;
```

Oracle

MERGE.sql (Fortführung)

```
-- step 1: iterate through local appearances referencing a geometry that
-- will be merged into the newly created gml:MultiSurface of the reference
-- building
for app_rec in app_cur loop
  if app_rec.id != app_id then
    app_id := app_rec.id;

    -- create a new appearance element for the reference building
    -- into which we are going to transfer the surface data
    execute immediate 'select appearance_seq.nextval from dual'
      into seq_val;
    execute immediate 'select id2 from match_overlap_relevant where id1=:1'
      into building_id using building_rec.building_id;

    execute immediate 'insert into appearance (id, name, name_codespace,
      description, theme, citymodel_id, cityobject_id)
      values (:1, null, null, :2, :3, null, :4)'
      using seq_val, app_rec.description, app_rec.theme, building_id;
  end if;

  -- move existing surface data into the newly created appearance
  if not_version_enabled then
    -- if appear_to_surface_data is not version-enabled
    -- a simple update does the job
    execute immediate 'update appear_to_surface_data
      set appearance_id=:1
      where appearance_id=:2 and surface_data_id=:3'
      using seq_val, app_rec.id, app_rec.sd_id;
  else
    -- if appear_to_surface_data is version-enabled
    -- updating is not possible since we are not allowed to change
    -- primary keys. so remove existing entry...
    execute immediate 'delete from appear_to_surface_data
      where appearance_id=:1 and surface_data_id=:2'
      using app_rec.id, app_rec.sd_id;

    -- ...and re-create it
    execute immediate 'insert into appear_to_surface_data
      (appearance_id, surface_data_id) values (:1, :2)'
      using seq_val, app_rec.sd_id;
  end if;
end loop;

-- step 2: if any surface data of the appearance references the root
-- element of the geometry to be merged we need to apply further checks
for geom_rec in geom_cur loop
  -- if just one geometry hierarchy has to be merged we simply let the
  -- textureparam point to the new root geometry element created for the
  -- reference building
  if building_rec.cnt_hierarchies = 1 then
    -- let textureparam point to newly created root element
    execute immediate 'update textureparam t
      set t.surface_geometry_id=(select container_id from
      merge_container_ids where building_id=:1)
      where t.surface_geometry_id=:2'
      using building_rec.building_id, geom_rec.hierarchy_id;
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
      -- step 1: iterate through local appearances referencing a geometry that will
      -- be merged into the newly created gml:MultiSurface of the reference building

      FOR app_rec IN app_cur LOOP
        IF app_rec.id != app_id THEN
          app_id := app_rec.id;

          -- create a new appearance element for the reference building
          -- into which we are going to transfer the surface data
          EXECUTE 'SELECT nextval(''appearance_id_seq'')' INTO seq_val;

          EXECUTE 'SELECT id2 FROM geodb_pkg.match_overlap_relevant WHERE id1=$1'
            INTO building_id USING building_rec.building_id;

          EXECUTE 'INSERT INTO appearance (id, name, name_codespace, description,
            theme, citymodel_id, cityobject_id)
            VALUES ($1, null, null, $2, $3, null, $4)'
              USING seq_val, app_rec.description, app_rec.theme, building_id;
        END IF;

        -- move existing surface data into the newly created appearance



          EXECUTE 'UPDATE appear_to_surface_data SET appearance_id=$1
          WHERE appearance_id=$2 AND surface_data_id=$3'
            USING seq_val, app_rec.id, app_rec.sd_id;

     -- Da keine Versionierung für die PostgreSQL/PostGIS-Version implementiert
     -- wurde, entfallen die Zeilen aus der Oracle-Version















      END LOOP;

      -- step 2: if any surface data of the appearance references the root element
      -- of the geometry to be merged we need to apply further checks
      FOR geom_rec IN geom_cur LOOP
        -- if just one geometry hierarchy has to be merged we simply let the
        -- textureparam point to the new root geometry element created for the
        -- reference building
        IF building_rec.cnt_hierarchies = 1 THEN
          -- let textureparam point to newly created root element
          EXECUTE 'UPDATE textureparam SET surface_geometry_id=
              (SELECT mci.container_id FROM geodb_pkg.merge_container_ids mci WHERE
               mci.building_id=$1) WHERE surface_geometry_id=$2'
            USING building_rec.building_id, geom_rec.hierarchy_id;
```

Oracle

MERGE.sql (Fortführung)

```
      -- copy gml:id to newly created root element - this is required
      -- for referencing the geometry from within the appearance
      execute immediate 'update surface_geometry s
        set (s.gmlid, s.gmlid_codespace)=(select gmlid, gmlid_codespace from
          surface_geometry where id=:1) where s.id=(select container_id from
            merge_container_ids where building_id=:2)'
              using geom_rec.hierarchy_id, building_rec.building_id;

    -- if more than one geometry hierarchy is merged into a single geometry
    -- hierarchy for the reference building, things are more complicated
    else
      declare
        counter number;
        gmlid surface_geometry.gmlid%type;
        gmlid_codespace surface_geometry.gmlid_codespace%type;
        cursor textureparam_cur is
          select * from textureparam
            where surface_data_id=geom_rec.sd_id
            and surface_geometry_id=geom_rec.hierarchy_id;
        cursor surface_geometry_cur is
          select * from surface_geometry where
            parent_id=geom_rec.hierarchy_id;
      begin
        begin
          execute immediate 'select gmlid, gmlid_codespace from
            surface_geometry where id=:1'
            into gmlid, gmlid_codespace using geom_rec.hierarchy_id;
        exception
          when others then
            gmlid := 'ID';
            gmlid_codespace := '';
        end;

        -- first we need to iterate over all textureparam which point to the
        -- root of the geometry hierachy to be merged.
        -- second we identify all direct childs of this root element. for
        -- each of these childs we create a copy of the original textureparam
        -- and let it point to the child.
        for textureparam_rec in textureparam_cur loop
          counter := 0;
          for surface_geometry_rec in surface_geometry_cur loop
            counter := counter + 1;

            -- create a new textureparam and let it point to the child
            -- instead of the root
            execute immediate 'insert into textureparam (surface_geometry_id,
              surface_data_id, is_texture_parametrization, world_to_texture,
              texture_coordinates) values (:1, :2, :3, :4, :5)'
              using surface_geometry_rec.id, geom_rec.sd_id,
                textureparam_rec.is_texture_parametrization,
                textureparam_rec.world_to_texture,
                textureparam_rec.texture_coordinates;
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
    -- copy gml:id to newly created root element - this is required
    -- for referencing the geometry from within the appearance
    EXECUTE 'UPDATE surface_geometry SET (gmlid, gmlid_codespace)=
        (SELECT s.gmlid, s.gmlid_codespace FROM surface_geometry s WHERE s.id=$1)
          WHERE id=(SELECT mci.container_id FROM geodb_pkg.merge_container_ids mci
            WHERE mci.building_id=$2)'
              USING geom_rec.hierarchy_id, building_rec.building_id;

    -- if more than one geometry hierarchy is merged into a single geometry
    -- hierarchy for the reference building, things are a bit more complicated
  ELSE
    DECLARE
      counter NUMERIC;
      gmlid surface_geometry.gmlid%TYPE;
      gmlid_codespace surface_geometry.gmlid_codespace%TYPE;
      textureparam_cur CURSOR FOR
        SELECT * FROM textureparam WHERE surface_data_id=geom_rec.sd_id
          AND surface_geometry_id=geom_rec.hierarchy_id;
      surface_geometry_cur CURSOR FOR
        SELECT * FROM surface_geometry WHERE parent_id=geom_rec.hierarchy_id;


    BEGIN
      BEGIN
        EXECUTE 'SELECT gmlid, gmlid_codespace FROM surface_geometry WHERE id=$1'
          INTO gmlid, gmlid_codespace USING geom_rec.hierarchy_id;

        EXCEPTION
          WHEN OTHERS THEN
            gmlid := 'ID';
            gmlid_codespace := '';
      END;

      -- first we need to iterate over all textureparam which point to the root of
      -- the geometry hierachy to be merged.
      -- second we identify all direct childs of this root element. for each of
      -- these childs we create a copy of the original textureparam and let it
      -- point to the child.
          FOR textureparam_rec IN textureparam_cur LOOP
            counter := 0;
            FOR surface_geometry_rec IN surface_geometry_cur LOOP
              counter := counter + 1;

              -- create a new textureparam and let it point to the child instead
              -- of the root
              EXECUTE 'INSERT INTO textureparam (surface_geometry_id,
                surface_data_id, is_texture_parametrization, world_to_texture,
                texture_coordinates) VALUES ($1, $2, $3, $4, $5)'
                  USING surface_geometry_rec.id, geom_rec.sd_id,
                  textureparam_rec.is_texture_parametrization,
                  textureparam_rec.world_to_texture,
                  textureparam_rec.texture_coordinates;
```

Oracle

MERGE.sql (Fortführung)

```
                  -- make sure the child geometry referenced by the textureparam
                  -- has a gml:id value
                  if surface_geometry_rec.gmlid is null then
                    execute immediate 'update surface_geometry
                      set gmlid=concat(:1, :2), gmlid_codespace=:3
                      where id=:4 and gmlid is null'
                        using gmlid, '_' || to_char(counter), gmlid_codespace,
                          surface_geometry_rec.id;
                  end if;
                end loop;
              end loop;
            end;
          end if;
        end loop;
      end;
    end loop;
  exception
      when others then
        dbms_output.put_line('move_appearance: ' || SQLERRM);
  end;


  procedure move_geometry
  is
  begin
    -- update parent of immediate children of all collected geometries
    execute immediate 'update surface_geometry s
      set s.parent_id = (select c.container_id from merge_container_ids c,
        merge_collect_geom g where s.parent_id = g.geometry_id and c.building_id =
          g.building_id)
            where s.parent_id in (select geometry_id from merge_collect_geom)';

    -- change nested solids into composite surfaces since we throw everything into
    -- a multisurface
    execute immediate 'update surface_geometry s
      set s.is_composite = 1,
      s.is_solid = 0
      where s.root_id in (select geometry_id from merge_collect_geom) and
        s.is_solid=1 and s.root_id<>s.id';

    -- update all root_ids
    execute immediate 'update surface_geometry s
      set s.root_id = (select c.container_id from merge_container_ids c,
        merge_collect_geom g
          where s.root_id = g.geometry_id and c.building_id = g.building_id)
            where s.root_id in (select geometry_id from merge_collect_geom) and
              s.root_id<>s.id';
  exception
      when others then
        dbms_output.put_line('move_geometry: ' || SQLERRM);
  end;
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
                  -- make sure the child geometry referenced by the textureparam has a
                  -- gml:id value
                  IF surface_geometry_rec.gmlid IS NULL THEN
                    EXECUTE 'UPDATE surface_geometry SET gmlid=concat($1, $2),
                      gmlid_codespace=$3 WHERE id=$4 AND gmlid is null'
                        USING gmlid, '_' || counter::VARCHAR, gmlid_codespace,
                          surface_geometry_rec.id;
                  END IF;
                END LOOP;
              END LOOP;
            END;
          END IF;
        END LOOP;
      END;
    END LOOP;

    EXCEPTION
      WHEN OTHERS THEN
        RAISE NOTICE 'move_appearance: %', SQLERRM;
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.merge_move_geometry()
RETURNS SETOF void AS $$
BEGIN
  -- UPDATE parent of immediate children of all collected geometries
  EXECUTE 'UPDATE surface_geometry SET parent_id =
    (SELECT c.container_id FROM geodb_pkg.merge_container_ids c, merge_collect_geom
       g WHERE parent_id = g.geometry_id AND c.building_id = g.building_id)
         WHERE parent_id IN (SELECT geometry_id FROM merge_collect_geom)';


  -- change nested solids into composite surfaces since we throw everything into a
  -- multisurface
  EXECUTE 'UPDATE surface_geometry SET is_composite = 1, is_solid = 0
    WHERE root_id IN (SELECT geometry_id FROM merge_collect_geom) AND is_solid=1 AND
      root_id != id';



  -- UPDATE all root_ids
  EXECUTE 'UPDATE surface_geometry SET root_id =
    (SELECT c.container_id FROM geodb_pkg.merge_container_ids c, merge_collect_geom
       g WHERE root_id = g.geometry_id AND c.building_id = g.building_id)
         WHERE root_id IN (SELECT geometry_id FROM merge_collect_geom) AND root_id !
           = id';
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'move_geometry: %', SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

MERGE.sql (Fortführung)

```
procedure update_lineage(lineage varchar2)
is
begin
  execute immediate 'update cityobject c
    set c.lineage = :1
    where c.id in (select b.id from building b, match_overlap_relevant m where
      b.building_root_id = m.id1)'
        using lineage;

    -- retrieve relevant building installation geometry
    execute immediate 'update cityobject c
      set c.lineage = :1
      where c.id in (select i.id from building_installation i, building b,
      match_overlap_relevant m
        where i.building_id = b.id
          and b.building_root_id = m.id1
          and i.is_external = 1)'
            using lineage;

    -- retrieve surfaces from relevant thematic surfaces
    execute immediate 'update cityobject c
    set c.lineage = :1
    where c.id in (select t.id from thematic_surface t, building b,
    match_overlap_relevant m
      where t.building_id = b.id
        and b.building_root_id = m.id1)'
          using lineage;

    -- retrieve all openings of all thematic surfaces beloning to all buildings
    -- and building parts
    execute immediate 'update cityobject c
    set c.lineage = :1
    where c.id in (select o.id from opening o, match_overlap_relevant m,
    thematic_surface t, building b, opening_to_them_surface ot
      where o.id = ot.opening_id
        and ot.thematic_surface_id = t.id
        and t.building_id = b.id
        and b.building_root_id = m.id1)'
          using lineage;

    -- room
    execute immediate 'update cityobject c
    set c.lineage = :1
    where c.id in (select r.id from room r, match_overlap_relevant m, building b
      where r.building_id = b.id
        and b.building_root_id = m.id1)'
          using lineage;

    -- building furniture (in rooms) --bei lod r in f geändert
    execute immediate 'update cityobject c
    set c.lineage = :1
    where c.id in (select f.id from building_furniture f, match_overlap_relevant
    m, room r, building b
      where f.room_id = r.id
        and r.building_id = b.id
            and b.building_root_id = m.id1)'
              using lineage;
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.merge_update_lineage(lineage VARCHAR)
RETURNS SETOF void AS $$
BEGIN
  EXECUTE 'UPDATE cityobject SET lineage = $1 WHERE id IN
    (SELECT b.id FROM building b, geodb_pkg.match_overlap_relevant m
       WHERE b.building_root_id = m.id1)'
         USING lineage;


  -- retrieve relevant building installation geometry
  EXECUTE 'UPDATE cityobject SET lineage = $1
    WHERE id IN
      (SELECT i.id FROM building_installation i, building b,
       geodb_pkg.match_overlap_relevant m
         WHERE i.building_id = b.id
           AND b.building_root_id = m.id1
           AND i.is_external = 1)'
             USING lineage;

  -- retrieve surfaces from relevant thematic surfaces
  EXECUTE 'UPDATE cityobject SET lineage = $1
    WHERE id IN
      (SELECT t.id FROM thematic_surface t, building b,
       geodb_pkg.match_overlap_relevant m
         WHERE t.building_id = b.id
           AND b.building_root_id = m.id1)'
             USING lineage;

  -- retrieve all openings of all thematic surfaces beloning to all buildings and
  -- building parts
  EXECUTE 'UPDATE cityobject SET lineage = $1
    WHERE id IN
      (SELECT o.id FROM opening o, geodb_pkg.match_overlap_relevant m,
       thematic_surface t, building b, opening_to_them_surface ot
         WHERE o.id = ot.opening_id
           AND ot.thematic_surface_id = t.id
           AND t.building_id = b.id
           AND b.building_root_id = m.id1)'
             USING lineage;

  -- room
  EXECUTE 'UPDATE cityobject SET lineage = $1
    WHERE id IN
      (SELECT r.id FROM room r, geodb_pkg.match_overlap_relevant m, building b
         WHERE r.building_id = b.id
           AND b.building_root_id = m.id1)'
             USING lineage;

  -- building furniture (in rooms) --if lod r is changed to f
  EXECUTE 'UPDATE cityobject SET lineage = $1
    WHERE id IN
      (SELECT f.id FROM building_furniture f, geodb_pkg.match_overlap_relevant m,
         room r, building b
         WHERE f.room_id = r.id
           AND r.building_id = b.id
           AND b.building_root_id = m.id1)'
             USING lineage;
```

Oracle

MERGE.sql (Fortführung)

```
    -- retrieve relevant internal (or external) building installation geometry
    -- (in rooms)
    execute immediate 'update cityobject c
    set c.lineage = :1
    where c.id in (select i.id from building_installation i,
    match_overlap_relevant m, building b, room r
      where i.room_id = r.id
        and r.building_id = b.id
                and b.building_root_id = m.id1)'
                  using lineage;

    -- retrieve surfaces from relevant thematic surfaces (in rooms)
    execute immediate 'update cityobject c
    set c.lineage = :1
      where c.id in (select t.id from thematic_surface t, match_overlap_relevant
        m, building b, room r
              where t.room_id = r.id
              and r.building_id = b.id
                and b.building_root_id = m.id1)'
    using lineage;

    -- retrieve all openings of all thematic surfaces beloning to all rooms in
    -- all buildings and building parts
    execute immediate 'update cityobject c
    set c.lineage = :1
    where c.id in (select o.id from opening o, match_overlap_relevant m,
    thematic_surface t, building b, opening_to_them_surface ot, room r
        where o.id = ot.opening_id
        and r.building_id = b.id
              and t.room_id = r.id
              and b.building_root_id = m.id1
              and ot.thematic_surface_id = t.id)'
    using lineage;

    -- retrieve relevant internal building installation geometry
    execute immediate 'update cityobject c
    set c.lineage = :1
    where c.id in (select i.id from building_installation i,
    match_overlap_relevant m, building b
      where i.building_id = b.id
        and b.building_root_id = m.id1
        and i.is_external = 0)'
    using lineage;
exception
    when others then
      dbms_output.put_line('collect_all_geometry: ' || SQLERRM);
end;
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
 -- retrieve relevant internal (or external) building installation geometry
 -- (in rooms)
  EXECUTE 'UPDATE cityobject SET lineage = $1
    WHERE id IN
      (SELECT i.id FROM building_installation i, geodb_pkg.match_overlap_relevant m,
       building b, room r
         WHERE i.room_id = r.id
           AND r.building_id = b.id
           AND b.building_root_id = m.id1)'
             USING lineage;

  -- retrieve surfaces from relevant thematic surfaces (in rooms)
  EXECUTE 'UPDATE cityobject SET lineage = $1
    WHERE id IN
      (SELECT t.id FROM thematic_surface t, geodb_pkg.match_overlap_relevant m,
       building b, room r
         WHERE t.room_id = r.id
           AND r.building_id = b.id
           AND b.building_root_id = m.id1)'
             USING lineage;

  -- retrieve all openings of all thematic surfaces beloning to all rooms in all
buildings and building parts
  EXECUTE 'UPDATE cityobject SET lineage = $1
    WHERE id IN
      (SELECT o.id FROM opening o, geodb_pkg.match_overlap_relevant m,
       thematic_surface t, building b, opening_to_them_surface ot, room r
         WHERE o.id = ot.opening_id
           AND r.building_id = b.id
           AND t.room_id = r.id
           AND b.building_root_id = m.id1
           AND ot.thematic_surface_id = t.id)'
             USING lineage;

  -- retrieve relevant internal building installation geometry
  EXECUTE 'UPDATE cityobject SET lineage = $1
    WHERE id IN
         (SELECT i.id FROM building_installation i, geodb_pkg.match_overlap_relevant
         m, building b
         WHERE i.building_id = b.id
           AND b.building_root_id = m.id1
           AND i.is_external = 0)'
             USING lineage;
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'collect_all_geometry: %', SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

MERGE.sql (Fortführung)

```
 procedure delete_head_of_merge_geometry

  is
    cursor geometry_cur is
      select geometry_id from merge_collect_geom;
  begin
    -- cleanly delete root of merged geometry hierarchy
    for geometry_rec in geometry_cur loop
      geodb_delete.delete_surface_geometry(geometry_rec.geometry_id, 0);
    end loop;

    geodb_delete.cleanup_appearances(0);
  end;


  procedure delete_relevant_candidates

  is
    cursor candidate_cur is
      select id1 from match_overlap_relevant;
  begin
    for candidate_rec in candidate_cur loop
      geodb_delete.delete_building(candidate_rec.id1);
    end loop;
  exception
    when others then
      dbms_output.put_line('delete_candidates: ' || SQLERRM);
  end;

END geodb_merge;
/
```

PostgreSQL / PostGIS

MERGE.sql (Fortführung)

```
CREATE OR REPLACE FUNCTION geodb_pkg.merge_delete_head_of_merge_geometry()
RETURNS SETOF void AS $$
DECLARE
  geometry_cur CURSOR FOR
    SELECT geometry_id FROM merge_collect_geom;
BEGIN
  -- cleanly delete root of merged geometry hierarchy
  FOR geometry_rec IN geometry_cur LOOP
    PERFORM geodb_pkg.del_delete_surface_geometry(geometry_rec.geometry_id);
  END LOOP;

  PERFORM geodb_pkg.del_cleanup_appearances(0);
END; $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION geodb_pkg.merge_delete_relevant_candidates()
RETURNS SETOF void AS $$
DECLARE
  candidate_cur CURSOR FOR
    SELECT id1 FROM geodb_pkg.match_overlap_relevant;
BEGIN
  FOR candidate_rec IN candidate_cur LOOP
    PERFORM geodb_pkg.del_delete_building(candidate_rec.id1);
  END LOOP;
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'delete_candidates: %', SQLERRM;
END; $$ LANGUAGE plpgsql;
```

Oracle

STAT.sql

```
/******************************************************************
* PACKAGE geodb_stat
*
* utility methods for creating database statistics
******************************************************************/
CREATE OR REPLACE PACKAGE geodb_stat
AS
  FUNCTION table_contents RETURN STRARRAY;
END geodb_stat;
/

CREATE OR REPLACE PACKAGE BODY geodb_stat
AS

  /******************************************************************
  * versioning_status
  *
  ******************************************************************/
  FUNCTION table_contents RETURN STRARRAY
  IS
    report STRARRAY := STRARRAY();
    ws VARCHAR2(30);
    cnt NUMBER;
    refreshDate DATE;
    reportDate DATE;
    pa_id PLANNING_ALTERNATIVE.ID%TYPE;
    pa_title PLANNING_ALTERNATIVE.TITLE%TYPE;

  BEGIN
    SELECT SYSDATE INTO reportDate FROM DUAL;
    report.extend; report(report.count) := ('Database Report on 3D City Model -
      Report date: ' || TO_CHAR(reportDate, 'DD.MM.YYYY HH24:MI:SS'));
    report.extend; report(report.count) :=
      ('===================================================================');

    -- Determine current workspace
    ws := DBMS_WM.GetWorkspace;
    report.extend; report(report.count) := ('Current workspace: ' || ws);

    IF ws != 'LIVE' THEN
      -- Get associated planning alternative
      SELECT id,title INTO pa_id,pa_title FROM PLANNING_ALTERNATIVE
      WHERE workspace_name=ws;
      report.extend; report(report.count) := ('  (PlanningAlternative ID ' || pa_id
        ||': "' || pa_title || '")');

      -- Query date of last refresh
      SELECT createtime INTO refreshDate
      FROM all_workspace_savepoints
      WHERE savepoint='refreshed' AND workspace=ws;
      report.extend; report(report.count) := ('Last refresh from LIVE workspace: '
        || TO_CHAR(refreshDate, 'DD.MM.YYYY HH24:MI:SS'));
    END IF;
    report.extend; report(report.count) := '';

    SELECT count(*) INTO cnt FROM citymodel;
    report.extend; report(report.count) := ('#CITYMODEL:\t\t\t' || cnt);
    . . . -- es folgen die restlichen Tabellen
```

PostgreSQL / PostGIS

STAT.sql

```
CREATE OR REPLACE FUNCTION geodb_pkg.stat_table_contents() RETURNS text[] AS $$
DECLARE
  report text[] = '{}';
  cnt NUMERIC;
  reportDate TIMESTAMP;
  -- Die restlichen Variablen waren nur für jene Zeilen notwendig, welche die
  -- Versionierung in Oracle betrafen.

BEGIN
  reportDate := NOW();
  report := array_append(report, 'Database Report on 3D City Model - Report date: '
    || TO_CHAR(reportDate, 'DD.MM.YYYY HH24:MI:SS'));
  report := array_append(report,
    '===================================================================');

  -- Da keine Versionierung für die PostgreSQL/PostGIS-Version implementiert
  -- wurde, entfallen die Zeilen aus der Oracle-Version.

  PERFORM array_append(report, '');

  SELECT COUNT(*) INTO cnt FROM citymodel;
  report := array_append(report, ('#CITYMODEL:\t\t\t' || cnt));
  . . . -- es folgen die restlichen Tabellen
```

Oracle

UTIL.sql

```
/******************************************************************
* TYPE STRARRAY
*
* global type for arrays of strings, e.g. used for log messages and reports
******************************************************************/
set term off;
set serveroutput off;

CREATE OR REPLACE TYPE STRARRAY IS TABLE OF VARCHAR2(32767);
/

DROP TYPE DB_INFO_TABLE;
CREATE OR REPLACE TYPE DB_INFO_OBJ AS OBJECT(
  SRID NUMBER,
  GML_SRS_NAME VARCHAR2(1000),
  COORD_REF_SYS_NAME VARCHAR2(80),
  COORD_REF_SYS_KIND VARCHAR2(24),
  VERSIONING VARCHAR2(100)
);
/

CREATE OR REPLACE TYPE DB_INFO_TABLE IS TABLE OF DB_INFO_OBJ;
/

/******************************************************************
* PACKAGE geodb_util
*
* utility methods for applications and subpackages
******************************************************************/
CREATE OR REPLACE PACKAGE geodb_util
AS
  FUNCTION versioning_table(table_name VARCHAR2) RETURN VARCHAR2;
  FUNCTION versioning_db RETURN VARCHAR2;
  . . .
END geodb_util;
/

CREATE OR REPLACE PACKAGE BODY geodb_util
AS

  /******************************************************************
  * versioning_table
  *
  * @param table_name name of the unversioned table, i.e., omit
  *                   suffixes such as _LT
  * @return VARCHAR2 'ON' for version-enabled, 'OFF' otherwise
  ******************************************************************/
  FUNCTION versioning_table(table_name VARCHAR2) RETURN VARCHAR2
  IS
    status USER_TABLES.STATUS%TYPE;
  BEGIN
    execute immediate 'SELECT STATUS FROM USER_TABLES WHERE TABLE_NAME=:1'
      into status using table_name || '_LT';
    RETURN 'ON';
  EXCEPTION
    WHEN others THEN
      RETURN 'OFF';
  END;
```

PostgreSQL / PostGIS

UTIL.sql

```
-- Der benutzerdefinierte Datentyp STRARRAY wurde nicht portiert.
-- Stattdessen wurde der text-Datentyp von PostgreSQL verwendet (ggf. als Array[])

-- Die Typen DB_INFO_OBJ und DB_INFO_TABLE wurden nur für die Funktion
-- db_metadata() verwendet und konnten dort bei der Portierung entfallen.
-- (siehe nächste Seite). Dementsprechend sind die Definitionen an dieser Stelle
-- nicht mehr notwendig.
```

```
/******************************************************************
* versioning_table
*
* @param table_name name of the unversioned table, i.e., omit
*                   suffixes such as _LT
* @RETURN VARCHAR 'ON' for version-enabled, 'OFF' otherwise
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_versioning_table(table_name VARCHAR)
RETURNS VARCHAR AS $$

BEGIN
  -- Diese Funktion könnte im Grunde genommen entfallen. Sie wurde beibehalten, um
  -- die API der 3DCityDB stabil zu halten.

  RETURN 'OFF';


END; $$ LANGUAGE plpgsql;
```

Oracle

UTIL.sql (Fortführung)

```
/******************************************************************
* versioning_db
*
* @return VARCHAR2 'ON' for version-enabled, 'PARTLY' and 'OFF'
******************************************************************/
FUNCTION versioning_db RETURN VARCHAR2
IS
  table_names STRARRAY;
  is_versioned BOOLEAN := FALSE;
  not_versioned BOOLEAN := FALSE;
BEGIN
  table_names := split('ADDRESS,. . .'); -- es folgen alle weiteren Tabellen

  FOR i IN table_names.first .. table_names.last LOOP
    IF versioning_table(table_names(i)) = 'ON' THEN
      is_versioned := TRUE;
    ELSE
      not_versioned := TRUE;
    END IF;

  END LOOP;

  IF is_versioned AND NOT not_versioned THEN
    RETURN 'ON';
  ELSIF is_versioned AND not_versioned THEN
    RETURN 'PARTLY';
  ELSE
    RETURN 'OFF';
  END IF;
END;

/******************************************************************
* db_info
*
* @param srid database srid
* @param srs database srs name
* @param versioning database versioning
******************************************************************/
PROCEDURE db_info(srid OUT DATABASE_SRS.SRID%TYPE, srs OUT DATABASE_SRS.GML_SRS_NAME
%TYPE, versioning OUT VARCHAR2)
IS
BEGIN
  execute immediate 'SELECT SRID, GML_SRS_NAME from DATABASE_SRS' into srid, srs;
  versioning := versioning_db;
END;

/******************************************************************
* db_metadata
*
******************************************************************/
FUNCTION db_metadata RETURN DB_INFO_TABLE
IS
  info_ret DB_INFO_TABLE;
  info_tmp DB_INFO_OBJ;
BEGIN
  info_ret := DB_INFO_TABLE();
  info_ret.extend;
```

PostgreSQL / PostGIS

UTIL.sql (Fortführung)

```
/******************************************************************
* versioning_db
*
* @RETURN VARCHAR 'ON' for version-enabled, 'PARTLY' and 'OFF'
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_versioning_db() RETURNS VARCHAR AS $$

-- Diese Funktion könnte im Grunde genommen entfallen. Sie wurde beibehalten, um
-- die API der 3DCityDB stabil zu halten.

BEGIN

    RETURN 'OFF';

END; $$ LANGUAGE plpgsql;

/******************************************************************
* db_info
*
* @param srid database srid
* @param srs database srs name
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_db_info(OUT srid DATABASE_SRS.SRID%TYPE,
OUT srs DATABASE_SRS.GML_SRS_NAME%TYPE)
RETURNS SETOF record AS $$

    SELECT srid, gml_srs_name FROM database_srs;

$$
LANGUAGE sql;

/*******************************************************************
* db_metadata
*
* @RETURN TABLE with columns SRID, GML_SRS_NAME, COORD_REF_SYS_NAME,
COORD_REF_SYS_KIND
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_db_metadata()
RETURNS TABLE(srid INTEGER, gml_srs_name VARCHAR(1000), coord_ref_sys_name
VARCHAR(2048), coord_ref_sys_kind VARCHAR(2048)) AS $$
-- Statt einer verschachtelten Tabelle, die in PostgreSQL nicht existiert, gibt die
-- Funktion eine Tabelle zurück. Dieses Konzept kann vom Importer/Exporter ebenso
-- gut verarbeitet werden.
BEGIN
```

Oracle

UTIL.sql (Fortführung)

```
    info_tmp := DB_INFO_OBJ(0, NULL, NULL, 0, NULL);

    execute immediate 'SELECT SRID, GML_SRS_NAME from DATABASE_SRS'
      into info_tmp.srid, info_tmp.gml_srs_name;
    execute immediate 'SELECT COORD_REF_SYS_NAME, COORD_REF_SYS_KIND from
      SDO_COORD_REF_SYS where SRID=:1' into info_tmp.coord_ref_sys_name,
        info_tmp.coord_ref_sys_kind using info_tmp.srid;
    info_tmp.versioning := versioning_db;
    info_ret(info_ret.count) := info_tmp;
    return info_ret;
  END;

  /*****************************************************************
  * error_msg
  *
  * @param err_code Oracle SQL error code, usually starting with '-',
  *                e.g. '-06404'
  * @return VARCHAR2 corresponding Oracle SQL error message
  ******************************************************************/
  FUNCTION error_msg(err_code VARCHAR2) RETURN VARCHAR2
  IS
  BEGIN

    RETURN SQLERRM(err_code);
    -- Die Idee der Funktion lies sich in PostgreSQL nicht umsetzen.
    -- In der portierten Version erfüllt die Funktion nicht den vorgesehenen
    -- Zweck, für eine stabile API der 3DCityDB wurde sie aber beibehalten
  END;

  /*****************************************************************
  * split
  *
  * @param list string to be splitted
  * @param delim delimiter used for splitting, defaults to ','
  * @return STRARRAY array of strings containing split tokens
  ******************************************************************/
  FUNCTION split(list VARCHAR2, delim VARCHAR2 := ',') RETURN STRARRAY
  . . .

  /*****************************************************************
  * min
  *
  * @param a first number value
  * @param b second number value
  * @return NUMBER the smaller of the two input number values
  ******************************************************************/
  FUNCTION min(a NUMBER, b NUMBER) RETURN NUMBER
  IS
  BEGIN
    IF a < b THEN
      RETURN a;
    ELSE
      RETURN b;
    END IF;
  END;
```

PostgreSQL / PostGIS

UTIL.sql (Fortführung)

```
    EXECUTE 'SELECT SRID, GML_SRS_NAME FROM DATABASE_SRS' INTO srid, gml_srs_name;
    EXECUTE 'SELECT srtext, srtext FROM spatial_ref_sys WHERE SRID=' || srid || ''
      INTO coord_ref_sys_name, coord_ref_sys_kind;
    -- Da die gesuchten Informationen zum Referenzsystem alle in einem Spaltenwert
    -- enthalten sind, müssen etwas umständliche String-Methoden verwendet werden.
    coord_ref_sys_name := split_part(coord_ref_sys_name, '"', 2);
    coord_ref_sys_kind := split_part(coord_ref_sys_kind, '[', 1);
    RETURN NEXT;
END; $$ LANGUAGE plpgsql;

/******************************************************************
* error_msg
*
* @param err_code PostgreSQL error code, e.g. '06404'
* @RETURN TEXT corresponding PostgreSQL error message
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_error_msg(err_code VARCHAR)
RETURNS TEXT AS $$
BEGIN
    BEGIN
        RAISE EXCEPTION USING ERRCODE = err_code;
    EXCEPTION
        WHEN OTHERS THEN
            RETURN SQLERRM;
   END;
END; $$ LANGUAGE plpgsql;


-- split wurde in der Funktion versioning_db eingesetzt. Da deren Inhalt bei der
-- Übersetzung auf ein Minimum reduziert wurde, wird split nicht mehr benötigt.
-- Darüber hinaus könnte der Zweck auch durch die PostgreSQL Funktion
-- string_to_array(text, text) erfüllt werden.




/******************************************************************
* min
*
* @param a first NUMERIC value
* @param b second NUMERIC value
* @RETURN NUMERIC the smaller of the two input NUMERIC values
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_min(a NUMERIC, b NUMERIC)
RETURNS NUMERIC AS $$
BEGIN
    IF a < b THEN
        RETURN a;
    ELSE
        RETURN b;
    END IF;
END; $$ LANGUAGE plpgsql;
```

Oracle

UTIL.sql (Fortführung)

```
/*****************************************************************
* transform_or_null
*
* @param geom the geometry whose representation is to be transformed using
* another coordinate system
* @param srid the SRID of the coordinate system to be used for the
* transformation.
* @return MDSYS.SDO_GEOMETRY the transformed geometry representation
******************************************************************/
FUNCTION transform_or_null(geom MDSYS.SDO_GEOMETRY, srid number)
RETURN MDSYS.SDO_GEOMETRY
IS
BEGIN
  IF geom is not NULL THEN
    RETURN SDO_CS.TRANSFORM(geom, srid);
  ELSE
    RETURN NULL;
  END IF;
END;

/*****************************************************************
* is_coord_ref_sys_3d
*
* @param srid the SRID of the coordinate system to be checked
* @return NUMBER the boolean result encoded as number: 0 = false, 1 = true
******************************************************************/
FUNCTION is_coord_ref_sys_3d(srid NUMBER) RETURN NUMBER
IS
  is_3d number := 0;
BEGIN
  execute immediate 'SELECT COUNT(*) from SDO_CRS_COMPOUND where SRID=:1'
    into is_3d using srid;
  if is_3d = 0 then
    execute immediate 'SELECT COUNT(*) from SDO_CRS_GEOGRAPHIC3D where SRID=:1'
      into is_3d using srid;
  end if;

  return is_3d;
END;

/*****************************************************************
* is_db_coord_ref_sys_3d
*
* @return NUMBER the boolean result encoded as number: 0 = false, 1 = true
******************************************************************/
FUNCTION is_db_coord_ref_sys_3d RETURN NUMBER
IS
  srid number;
BEGIN
  execute immediate 'SELECT srid from DATABASE_SRS' into srid;
  return is_coord_ref_sys_3d(srid);
END;

/*
* code taken from http://forums.oracle.com/forums/thread.jspa?
* messageID=960492&#960492
*/
function to_2d (geom mdsys.sdo_geometry, srid number)
```

PostgreSQL / PostGIS

UTIL.sql (Fortführung)

```
/******************************************************************
* transform_or_null
*
* @param geom the geometry whose representation is to be transformed using another
* coordinate system
* @param srid the SRID of the coordinate system to be used for the transformation.
* @RETURN GEOMETRY the transformed geometry representation
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_transform_or_null(geom GEOMETRY, srid
INTEGER) RETURNS geometry AS $$
BEGIN
    IF geom IS NOT NULL THEN
        RETURN ST_Transform(geom, srid);
    ELSE
        RETURN NULL;
    END IF;
END; $$ LANGUAGE plpgsql;

/******************************************************************
* is_coord_ref_sys_3d
*
* no 3D-Coord.-Reference-System defined in the spatial_ref_sys-table of PostGIS 2.0
* by default refer to spatialreference.org for INSERT-statements of 3D-SRIDs
* they can be identified by the AXIS UP in the srtext
*
* @param srid the SRID of the coordinate system to be checked
* @RETURN NUMERIC the boolean result encoded as NUMERIC: 0 = false, 1 = true
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_is_coord_ref_sys_3d(srid INTEGER) RETURNS
INTEGER AS $$
  DECLARE
     is_3d INTEGER := 0;
  BEGIN
     EXECUTE 'SELECT count(*) FROM spatial_ref_sys WHERE auth_srid=$1 AND srtext
       LIKE ''%UP]%''' INTO is_3d USING srid;


    RETURN is_3d;
  END; $$ LANGUAGE plpgsql;

/******************************************************************
* is_db_coord_ref_sys_3d
*
* @RETURN NUMERIC the boolean result encoded as NUMERIC: 0 = false, 1 = true
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_is_db_coord_ref_sys_3d()
RETURNS INTEGER AS $$
  DECLARE
    srid INTEGER;
  BEGIN
    EXECUTE 'SELECT srid from DATABASE_SRS' INTO srid;
    RETURN geodb_pkg.util_is_coord_ref_sys_3d(srid);
  END; $$ LANGUAGE plpgsql;

-- Die Funktion to_2d existiert in PostGIS unter dem Namen ST_Force_2D(geometry)
```

PostgreSQL / PostGIS

UTIL.sql (Fortführung)

```
-- Die folgenden Funktionen wurden zusätzlich für die Portierung realisiert.
-- Es handelt sich dabei nur um weitere Hilfsfunktionen, die unabhängig vom
-- Importer/Exporter sind.

/******************************************************************
* change_db_srid
*
* Changes the database-SRID, if wrong SRID was used for CREATE_DB.
* Also helpful if using an instance of the 3DCityDB as a template database.
* It should only be executed on an empty database to avoid any errors.
*
* @param db_srid the SRID of the coordinate system to be further used in the
* database
* @param db_gml_srs_name the GML_SRS_NAME of the coordinate system to be further
* used in the database
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_change_db_srid (db_srid INTEGER,
db_gml_srs_name VARCHAR) RETURNS SETOF void AS $$
BEGIN
  -- Drop spatial indexes
  DROP INDEX CITYOBJECT_SPX;
  . . .

  -- Drop geometry columns from tables and geometry_columns-view
  PERFORM DropGeometryColumn('cityobject', 'envelope');
  . . .

  -- Update entry in DATABASE_SRS-table
  UPDATE DATABASE_SRS SET SRID=db_srid, GML_SRS_NAME=db_gml_srs_name;

  -- Create geometry columns in associted tables and add them to geometry_columns-
  -- view again
  PERFORM AddGeometryColumn('cityobject', 'envelope', db_srid, 'POLYGON', 3);
  . . .

  -- Create spatial indexes
  CREATE INDEX CITYOBJECT_SPX ON CITYOBJECT
    USING GIST ( ENVELOPE gist_geometry_ops_nd );
  . . .
END; $$ LANGUAGE plpgsql;


/******************************************************************
* on_delete_action
*
* Removes a contraint to add it again with parameters
* ON UPDATE CASCADE ON DELETE CASCADE or RESTRICT
*
* @param table_name defines the table to which the constraint belongs to
* @param fkey_name name of the foreign key that is updated
* @param column_name defines the column the constraint is relying on
* @param ref_table
* @param ref_column
* @param action whether CASCADE (default) or RESTRICT
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_on_delete_action(
  table_name VARCHAR,
  fkey_name VARCHAR,
```

PostgreSQL / PostGIS

UTIL.sql (Fortführung)

```
  column_name VARCHAR,
  ref_table VARCHAR,
  ref_column VARCHAR,
  action VARCHAR)
RETURNS SETOF void AS
$$
BEGIN
  EXECUTE 'ALTER TABLE ' || table_name || ' DROP CONSTRAINT ' || fkey_name ||
    ', ADD CONSTRAINT ' || fkey_name || ' FOREIGN KEY (' || column_name || ')
      ' || 'REFERENCES ' || ref_table || '(' || ref_column || ') ' ||
        'ON UPDATE CASCADE ON DELETE ' || action;

  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'Error on constraint %: %', fkey_name, SQLERRM;
END;
$$
LANGUAGE plpgsql;


/******************************************************************
* update_constraints
*
* uses the FUNCTION on_delete_action for updating all the contraints
*
* @param action whether CASCADE (default) or RESTRICT
******************************************************************/
CREATE OR REPLACE FUNCTION geodb_pkg.util_update_constraints(action VARCHAR DEFAULT
'CASCADE') RETURNS SETOF void AS $$
BEGIN
  IF action <> 'CASCADE' THEN
    action := 'RESTRICT';
    RAISE NOTICE 'Constraints are set to ON DELETE RESTRICT';
  END IF;

  PERFORM geodb_pkg.util_on_delete_action(
    'ADDRESS_TO_BUILDING','ADDRESS_TO_BUILDING_FK','BUILDING_ID','BUILDING',
      'ID',action);
  PERFORM geodb_pkg.util_on_delete_action(
    'ADDRESS_TO_BUILDING','ADDRESS_TO_BUILDING_ADDRESS_FK','ADDRESS_ID',
      'ADDRESS','ID',action);
  . . .
END; $$ LANGUAGE plpgsql;

-- Das Aktualisieren der Foreign Key Parameter zu ON DELETE CASCADE bewirkt beim
-- Entfernen eines Eintrages das gleichzeitige Löschen aller auf die Zeile
-- referenzierenden Inhalte in anderen Tabellen, damit der Bestand konsistent
-- bleibt. Die Funktion entstand bei Tests mit dem Versionierungs-Tool pgVersion.
-- Die Logik von pgVersion spielte Veränderungen im Datenbestand mit einem DELETE
-- und darauf folgenden INSERT ein, was einige Fremdschlüssel verletzt hätte. An
-- diesem Beispiel ist zu erkennen, das die Funktion zukünftig für Zwecke der
-- Fortführung oder Versionierung der Datenbank hilfreich sein kann.
```

Anhang B

Java Quellcode

Anders als bei den SQL- und PL/pgSQL-Skripten sind die Java-Klassen nicht im Installations-Paket sichtbar. Der Quellcode kann separat von der Projekt-Homepage heruntergeladen werden. Nichts desto trotz wird dieser Teil des Anhangs nur die wichtigsten Passagen der Anpassungen gegen die *Oracle*-Version vergleichen. Er entspricht damit der ebenfalls im Internet und im Release verfügbaren Dokumentation.

Die Übersichtsboxen beziehen sich jeweils auf den Inhalt des Unterkapitels. Sie sollen einen schnellen Überblick geben, welche Klassen angepasst werden mussten und auf welche Projektpakete sich diese verteilt haben.

Zeichenerklärung:

Pakete:

- ☐ (weiß) = keine Klasse musste in dem Paket angepasst werden
- ■ (gelb) = eine oder mehrere Klassen in dem Paket mussten inhaltlich verändert werden
- ■ (orange) = das Paket enthält Klassen, die ggf. zukünftig portiert werden müssten

Speicherort der Klasse:

[A]	from package api	[M cityC]	modules.citygml.common
[Cmd]	cmd	[M cityE]	modules.citygml.exporter
[C]	config	[M cityI]	modules.citygml.importer
[D]	database	[M com]	modules.common
[E]	event	[M db]	modules.database
[G]	gui	[M kml]	modules.kml
[L]	log	[M pref]	modules.prefrences
[P]	plugin	[oracle]	oracle.spatial.geometry
[U]	util		

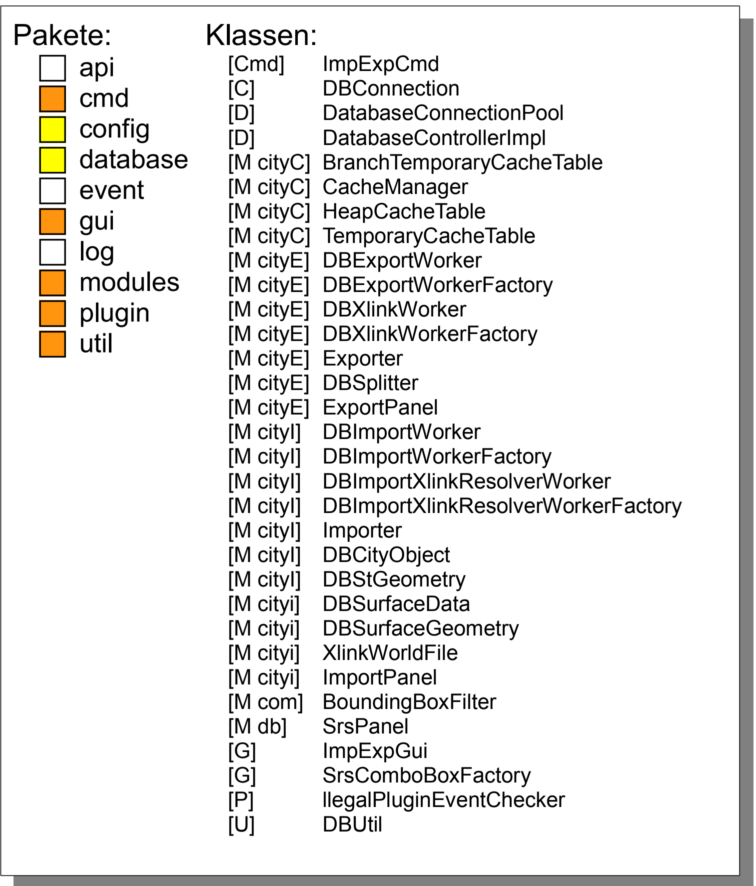
Quellcode:

59 Abweichungen starten ab Zeile 59 in der entsprechenden Klasse.
115+ Diese und folgende Zeilen waren nicht zu übersetzen.
wdh Das Quellcode-Beispiel wiederholt sich in der selben Klasse.
wdh+ Das Quellcode-Beispiel wiederholt sich in der selben und anderen Klassen.

```
//private Integer port = 1521;
private Integer port = 5432;
```

`//private Integer port = 1521;` auskommentierter *Oracle*-spezifischer Code
`private Integer port = 5432;` *PostGIS*-spezifischer Code

B.1. Verbindung zur Datenbank



Um eine Verbindung zu einer *PostgreSQL*-Datenbank herzustellen, bedarf es nicht vieler Änderungen, vorrausgesetzt man nutzt weiterhin den *Universal Connection Pool (UCP)* von Oracle. Die PoolDataSource des *UCP* muss auf eine Quelle („DataSource“) von *PostgreSQL* zugreifen können und eine URL den JDBC-Treiber von *PostgreSQL* adressieren. Wenn die Klasse `PGSimpleDataSource` dafür verwendet wird, ist es erforderlich den Datenbank-Namen separat zu definieren. Die Übergabe mittels `conn.getSid()` innerhalb einer URL wird nicht korrekt interpretiert, was möglicherweise an der unterschiedlichen Definition einer Datenbank zwischen *Oracle* und *PostgreSQL* liegt. Wenn über ein Netzwerk gearbeitet wird, ist auch das explizite Setzen des Server-Namens und der Port-Nummer notwendig.

Deshalb wurde im Endeffekt die Klasse `org.postgresql.Driver` verwendet, um eine Verbindungs-URL nutzen zu können Der Absatz rund um die Connection-Properties wurde ausgeklammert, da die *PostgreSQL*-eigene `Connection` Klasse nur über die gleichen Attribute wie die Java `Connection` Klasse verfügte (d.h. `CONNECTION_PROPERTY_USE_THREADLOCAL_BUFFER_CACHE` war nicht verfügbar).

Leider entspricht die Verwendung des *UCP*s nicht dem Anspruch einer vollständigen Umsetzung der *3DCityDB* auf Open-Source-Bestandteile, weshalb freie Alternativen (Apache *Jakarta DBCP* [www61], Apache *Tomcat 7* [www62] und *C3PO* [www63]) auf ihre Anwendbarkeit untersucht wurden. Da sich aber kein schneller Erfolg einstellen wollte und speziell Apache's *DBCP* bereits bei Tests der Entwickler ausgemustert wurde, wird nun weiterhin der *UCP* benutzt. Puristisch gesehen gehört auch Java selbst zu Oracle.

Die Übersichtbox verdeutlicht außerdem wie weit die Methoden der Klasse `DatabaseConnectionPool` verstreut sind. Oft handelt es sich zwar nur um die Methode `getConnection()`, dennoch wären Anpassungsversuche zeitaufwändig. Die betroffenen Klassen sind nicht in der Box aufgeführt, weil letztendlich keine Veränderungen stattfanden.

de.tub.citydb.config.project.database.**DBConnection**

```
59      //private Integer port = 1521;
        private Integer port = 5432;
```

de.tub.citydb.database.**DatabaseConnectionPool**

```
64      //private final String poolName = "oracle.pool";
        private final String poolName = "postgresql.pool";

115     // poolDataSource.setConnectionFactoryClassName(
        //     "oracle.jdbc.pool.OracleDataSource");
        //
        // poolDataSource.setURL("jdbc:oracle:thin:@//" + conn.getServer() +
              ":" + conn.getPort()+ "/" + conn.getSid());
        poolDataSource.setConnectionFactoryClassName("org.postgresql.Driver");
        poolDataSource.setURL("jdbc:postgresql://" + conn.getServer() + ":" +
               conn.getPort() + "/" + conn.getSid());

115+    // set connection properties
```

B.2. Aufruf von PL/pgSQL-Funktionen

Pakete:		Klasses:	
☐	api	[M db]	IndexOperation
☐	cmd	[M cityI]	Importer
☐	config	[M cityE]	DBAppearance
☐	database	[M cityE]	DBBuilding
☐	event	[M cityE]	DBBuildingFurniture
☐	gui	[M cityE]	DBCityFurniture
☐	log	[M cityE]	DBCityObject
■	modules	[M cityE]	DBCityObjectGroup
☐	plugin	[M cityE]	DBGeneralization
■	util	[M cityE]	DBGenericCityObject
		[M cityE]	DBReliefFeature
		[M cityE]	DBSolitaryVegetatObject
		[M cityE]	DBSurfaceGeometry
		[M cityE]	DBThematicSurface
		[M cityE]	DBTransportationComplex
		[M cityE]	DBWaterBody
		[U]	DBUtil

Viele Interaktionsmöglichkeiten mit dem *Importer/Exporter* rufen die im vorherigen Anhang ausführlich präsentierten PL/pgSQL-Datenbank-Prozeduren auf, d.h. auf Java-Seite muss nur der Funktionsname verändert werden (`geodb_pkg.prefix_name`).

B.2.1. Funktionen aus den Dateien IDX, UTIL und STAT

für alle de.tub.citydb.modules.citygml.exporter.database.content.**DB***

```
//geodb_util.transform_or_null(...
geodb_pkg.util_transform_or_null(...
```

de.tub.citydb.util.database.**DBUtil**

```
73      // private static OracleCallableStatement callableStmt;
        private static CallableStatement callableStmt;

91      // rs = stmt.executeQuery("select * from table(geodb_util.db_metadata)");
        rs = stmt.executeQuery("select * from geodb_pkg.util_db_metadata() as t");

199     // callableStmt = (OracleCallableStatement)conn.prepareCall("{? = call
wdh     //     geodb_stat.table_contents}");
        callableStmt = (CallableStatement)conn.prepareCall("{? = call
               geodb_pkg.stat_table_contents()}");

200     // callableStmt.registerOutParameter(1, OracleTypes.ARRAY, "STRARRAY");
wdh     callableStmt.registerOutParameter(1, Types.ARRAY);

203     // ARRAY result = callableStmt.getARRAY(1);
wdh     Array result = callableStmt.getArray(1);
```

```
374   // String call = type == DBIndexType.SPATIAL ?
wdh   //             "{? = call geodb_idx.drop_spatial_indexes}" :
      //                   "{? = call geodb_idx.drop_normal_indexes}";
      String call = type == DBIndexType.SPATIAL ?
            "{? = call geodb_pkg.idx_drop_spatial_indexes()}" :
                  "{? = call geodb_pkg.idx_drop_normal_indexes()}";
      // callableStmt = (OracleCallableStatement)conn.prepareCall(call);
      callableStmt = (CallableStatement)conn.prepareCall(call);
```

B.2.2. Das Berechnen der Bounding Box

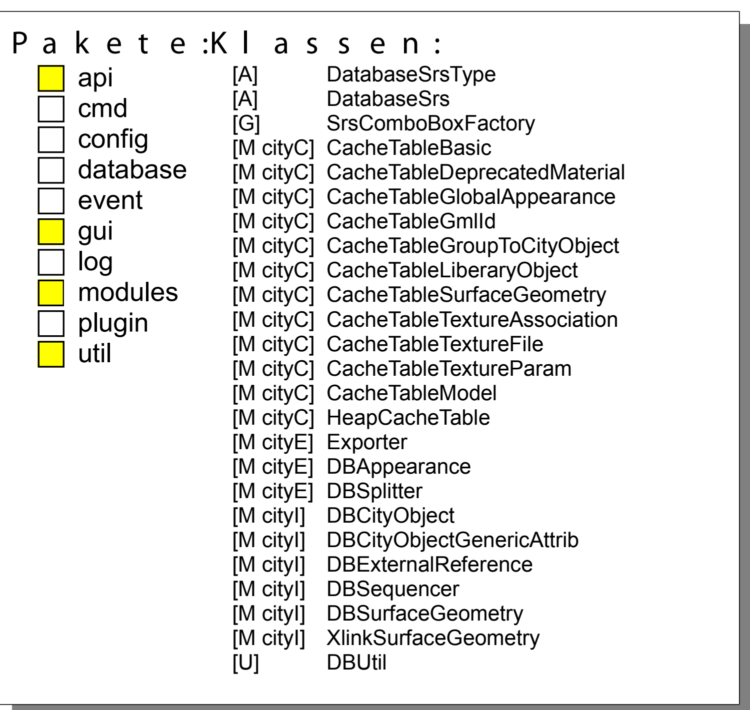
Für die Berechnung der Bounding Box mussten Variablen, die sich auf das Workspace Management von *Oracle* beziehen, auskommentiert werden. Die Datenbank-Abfrage enthielt *Oracle*-spezifische Funktionen, die mit äquivalenten *PostGIS* Operationen ausgetauscht werden mussten. Aus `sdo_aggr_mbr` wurde `ST_Extent` und aus `geodb_util.to2d` wurde `ST_Force_2d`. Da die Rückgabe von `ST_Extent` vom *PostGIS*-Datentyp BOX2D ist, war ein Typumwandlung („Cast") in den GEOMETRY Datentyp notwendig, damit die Geometrie in Java angesprochen werden konnte. Als Geometrie enthält das Rechteck jedoch 5 anstatt wie in *Oracle* 2 Punkte (Lower Left Bottom, Upper Right Top). Die erhaltenen Werte werden für das Konstruieren einer neuen Bounding Box verwendet, die transformierte Koordinaten erhält. Das Konstruieren mit `ST_GeomFromEWKT` funktionierte jedoch nicht, wenn die einzelnen Koordinaten in dem `PreparedStatement` als Platzhalter dienten. Der gesamte EWKT-String wurde daraufhin als Platzhalter gewählt.

de.tub.citydb.util.database.**DBUtil**

```
237   // public static BoundingBox calcBoundingBox(Workspace workspace,
      //     FeatureClassMode featureClass) throws SQLException {
      public static BoundingBox calcBoundingBox(FeatureClassMode featureClass)
            throws SQLException {

248   // String query = "select sdo_aggr_mbr(geodb_util.to_2d(
      //     ENVELOPE, (select srid from database_srs)))
      //             from CITYOBJECT where ENVELOPE is not NULL";
      String query = "select ST_Extent(ST_Force_2d(envelope))::geometry
            from cityobject where envelope is not null";

317   // double[] points = jGeom.getOrdinatesArray();
      // if (dim == 2) {
      //     xmin = points[0];
      //     ymin = points[1];
      //     xmax = points[2];
      //     ymax = points[3];
      // } else if (dim == 3) {
      //     xmin = points[0];
      //     ymin = points[1];
      //     xmax = points[3];
      //     ymax = points[4];
      // }
      xmin = (geom.getPoint(0).x);
      ymin = (geom.getPoint(0).y);
      xmax = (geom.getPoint(2).x);
      ymax = (geom.getPoint(2).y);
```

```
29      // psQuery = conn.prepareStatement("select SDO_CS.TRANSFORM(
        //     MDSYS.SDO_GEOMETRY(2003, " + sourceSrid + ", NULL,
        //     MDSYS.SDO_ELEM_INFO_ARRAY(1, 1003, 1), " +
        //     "MDSYS.SDO_ORDINATE_ARRAY(?,?,?,?)), " + targetSrid + ")from dual");
        // psQuery.setDouble(1, bbox.getLowerLeftCorner().getX());
        // psQuery.setDouble(2, bbox.getLowerLeftCorner().getY());
        // psQuery.setDouble(3, bbox.getUpperRightCorner().getX());
        // psQuery.setDouble(4, bbox.getUpperRightCorner().getY());
        psQuery = conn.prepareStatement("select ST_Transform(ST_GeomFromEWKT(?), "
              + targetSrid + ")");

boxGeom = "SRID=" + sourceSrid + ";POLYGON((" +
              bbox.getLowerLeftCorner().getX() + " " +

              bbox.getLowerLeftCorner().getY() + "," +
              bbox.getLowerLeftCorner().getX() + " " +
              bbox.getUpperRightCorner().getY() + "," +
              bbox.getUpperRightCorner().getX() + " " +
              bbox.getUpperRightCorner().getY() + "," +
              bbox.getUpperRightCorner().getX() + " " +
              bbox.getLowerLeftCorner().getY() + "," +
              bbox.getLowerLeftCorner().getX() + " " +
              bbox.getLowerLeftCorner().getY() + "))";

        psQuery.setString(1, boxGeom);

645     // double[] ordinatesArray = geom.getOrdinatesArray();
        // result.getLowerCorner().setX(ordinatesArray[0]);
        // result.getLowerCorner().setY(ordinatesArray[1]);
        // result.getUpperCorner().setX(ordinatesArray[2]);
        // result.getUpperCorner().setY(ordinatesArray[3]);
        result.getLowerLeftCorner().setX(geom.getPoint(0).x);
        result.getLowerLeftCorner().setY(geom.getPoint(0).y);
        result.getUpperRightCorner().setX(geom.getPoint(2).x);
        result.getUpperRightCorner().setY(geom.getPoint(2).y);
```

B.3. Datenbank-Spezifika in Java



B.3.1. Das Datenbank-SRS

PostGIS bietet bisher standardmäßig keine 3D-Refrenzsysteme an. Über die Website spatialreference.org [www64], lassen sich jedoch `INSERT`-Statements für einzelne Referenzsysteme abrufen. Leider werden geographische Bezugssysteme in 2D und 3D als GEOGCS klassifiziert. Die Typisierung muss sich aber unterscheiden, damit die Funktion `is3D()` in der Klasse `DBUtil` korrekt arbeitet. Diese orientiert sich wiederum an der Klasse `DatabaseSrsType`, wo ein dreidimensionales geographisches Referenzsystem als GEOGCS3D hart kodiert wurde. Dementsprechend müsste das `INSERT`-Statement von spatialreference.org angepasst werden, bevor es ausgeführt wird. Es ist generell unklar, ob eine mögliche zukünftige Unterstützung von 3D-SRIDs ähnlich stark reglementiert sein wird wie in *Oracle Spatial*, z.B. dass ein 3D-SRID nur mit einem 3D-Index kommunizieren kann etc. Ggf. wären bisherige Überprüfungen der Dimension wie für die *Oracle*-Version nicht von Nöten.

Weiterhin sind die Unterschiede zwischen den Tabelln `SDO_COORD_REF_SYS` (*Oracle Spatial*) und `SPATIAL_REF_SYS` (*PostGIS*) zu beachten. Ein Großteil der Informationen über das SRID steht bei der *PostGIS* Tabelle in der Spalte `srtext` und kann nur über etwas umständliche String-Methoden extrahiert werden.

de.tub.citydb.api.database.**DatabaseSrsType**

```
4       PROJECTED("PROJCS", "Projected"),
        GEOGRAPHIC2D("GEOGCS", "Geographic2D"),
        GEOCENTRIC("GEOCCS", "Geocentric"),
        VERTICAL("VERT_CS", "Vertical"),
        ENGINEERING("LOCAL_CS", "Engineering"),
        COMPOUND("COMPD_CS", "Compound"),
        GEOGENTRIC("n/a", "Geogentric"),
        GEOGRAPHIC3D("GEOGCS3D", "Geographic3D"),
        UNKNOWN("", "n/a");
```

de.tub.citydb.util.database.**DBUtil**

```
141     // psQuery = conn.prepareStatement("select coord_ref_sys_name,
        //     coord_ref_sys_kind from sdo_coord_ref_sys where srid = ?");
        psQuery = conn.prepareStatement("select split_part(srtext, '\"', 2) as
        coord_ref_sys_name, split_part(srtext, '[', 1) as coord_ref_sys_kind
        FROM spatial_ref_sys WHERE SRID = ? ");

704     // psQuery = conn.prepareStatement(srs.getType() ==
        //     DatabaseSrsType.GEOGRAPHIC3D ?
        //     "select min(crs2d.srid) from sdo_coord_ref_sys crs3d,
        //     sdo_coord_ref_sys  crs2d where crs3d.srid = " + srs.getSrid() +
        //     " and  crs2d.coord_ref_sys_kind = 'GEOGRAPHIC2D'
        //     and crs3d.datum_id = crs2d.datum_id" :
        //           "select cmpd_horiz_srid from sdo_coord_ref_sys
        //           where srid = " + srs.getSrid());
        psQuery = conn.prepareStatement(srs.getType()== DatabaseSrsType.COMPOUND ?
         "select split_part((split_part(srtext,'AUTHORITY[\"EPSG\",\"',5)),'\"',1)
              from spatial_ref_sys where auth_srid = " + srs.getSrid() :
            // searching 2D equivalent for 3D SRID
         "select min(crs2d.auth_srid) from spatial_ref_sys crs3d, spatial_ref_sys
              crs2d where (crs3d.auth_srid = " + srs.getSrid() + " and split_part
                     (crs3d.srtext, '[', 1) LIKE 'GEOGCS' AND
                                 split_part(crs2d.srtext, '[', 1) LIKE 'GEOGCS' " +
              //do they have the same Datum_ID?
              "and split_part(
                      (split_part(crs3d.srtext,'AUTHORITY[\"EPSG\",\"',3)),'\"',1)
              = split_part(
                     (split_part(crs2d.srtext,'AUTHORITY[\"EPSG\",\"',3)),'\"',1))
              OR " +
              // if srtext has been changed for Geographic3D
              "(crs3d.auth_srid = " + srs.getSrid() + " " and
                     split_part(crs3d.srtext, '[', 1) LIKE 'GEOGCS3D' AND
                                 split_part(crs2d.srtext, '[', 1) LIKE 'GEOGCS' " +
              //do they have the same Datum_ID?
              "and split_part(
                     (split_part(crs3d.srtext,'AUTHORITY[\"EPSG\",\"',3)),'\"',1)
              = split_part(
                  (split_part(crs2d.srtext,'AUTHORITY[\"EPSG\",\"',3)),'\"',1))");
```

B.3.2. Bounding Box Filter und Optimizer Hints

In der Klasse `DBSplitter.java` werden die Daten bei aktivierter Bounding Box Filterung nach ihrer räumlichen Lage zum Filter-Rechteck abgefragt. In *Oracle* funktioniert das mit `SDO_RELATE`, in *PostGIS* mit `ST_Relate`. Beide Funktionen überprüfen die topologische Beziehungen anhand einer 9er-Intersektions-Matrix (DE-9IM)

nach [EGENHOFER, HERRING 1990] und [CLEMENTINI, DI FELICE 1994]. In *Oracle* lässt sich das Maskenattribut in SDO_RELATE mittels logischem Oder (+) kombinieren. Die Form bleibt dabei ein String. Das geht in *PostGIS* nicht, weil `ST_Relate` nur eine Maske erlaubt. Die Schreibweise entspricht der DE-9IM. Die erste Zelle des `bboxFilter`-Arrays beinhaltet daher vier Abfragen, die jeweils mit „or" verbunden sind.

Eine weitere Besonderheit, die nur die Klasse `DB_Splitter` betrifft, ist die Verwendung von Optimizer Hints. Sie werden eingesetzt, um dem Abfrage-Optimizer Entscheidungen bei der Wahl eines optimalen Ausführungsplans abzunehmen. Da Optimizer Hints in *PostgreSQL* jedoch nicht eingesetzt werden, musste jedes Auftreten im Quellcode ausgeklammert werden.

de.tub.citydb.modules.citygml.exporter.database.content.**DBSplitter**

```
168   //     String filter = "SDO_RELATE(co.ENVELOPE, MDSYS.SDO_GEOMETRY(2003, "
      //            + bboxSrid + ", NULL, " +
      //            "MDSYS.SDO_ELEM_INFO_ARRAY(1, 1003, 3), " +
      //            "MDSYS.SDO_ORDINATE_ARRAY(" + minX + ", " + minY + ", " + maxX
      //            + ", " + maxY + ")), 'mask=";
      //            bboxFilter[0] = filter + "inside+coveredby') = 'TRUE'";
      //            bboxFilter[1] = filter + "equal') = 'TRUE'";

      //     if (overlap)
      //            bboxFilter[2] = filter + "overlapbdyintersect') = 'TRUE'";
      String filter = "ST_Relate(co.ENVELOPE, " +
                            "ST_GeomFromEWKT('SRID=" + bboxSrid + ";POLYGON((" +
                              minX + " " + minY + "," +
                              minX + " " + maxY + "," +
                              maxX + " " + maxY + "," +
                              maxX + " " + minY + "," +
                              minX + " " + minY + "))'), ";

      bboxFilter[0] = "(" + filter + "'T*F**F***') = 'TRUE' or " + // inside
                                filter + "'*TF**F***') = 'TRUE' or " + // coveredby
                                filter + "'**FT*F***') = 'TRUE' or " + // coveredby
                                filter + "'**F*TF***') = 'TRUE')"; // coveredby
      bboxFilter[1] = filter + "'T*F**FFF*') = 'TRUE'";    // equal

      if (overlap)
             bboxFilter[2] = filter + "'T*T***T**') = 'TRUE'"; //overlapbdyinter.
```

B.3.3. Abfragen beim Import

Einige Abfragen der `Importer` Klassen verwenden *Oracle*-spezifische Methoden.

de.tub.citydb.modules.citygml.importer.database.content.**DBCityObject**

```
134   // SYSDATE
      now()
```

de.tub.citydb.modules.citygml.importer.database.content.**DBCityObjectGenericAttrib**

```
63      // CITYOBJECT_GENERICATT_SEQ.nextval
        nextval('CITYOBJECT_GENERICATTRIB_ID_SEQ')
```

de.tub.citydb.modules.citygml.importer.database.content.**DBExternalReference**

```
58      // EXTERNAL_REF_SEQ.nextval
        nextval('EXTERNAL_REFERENCE_ID_SEQ')
```

de.tub.citydb.modules.citygml.importer.database.content.**DBSequencer**

```
53      // pstsmt = conn.prepareStatement("select " + sequence.toString() +
               ".nextval from dual");
        pstsmt = conn.prepareStatement("select nextval('" + sequence.toString() +
               "')");
```

de.tub.citydb.modules.citygml.importer.database.xlink.resolver.**XlinkSurfaceGeometry**

```
92      // psSelectSurfGeom = batchConn.prepareStatement("select sg.*, LEVEL from
               SURFACE_GEOMETRY sg start with sg.ID=? connect by prior
                      sg.ID=sg.PARENT_ID");
        psSelectSurfGeom = batchConn.prepareStatement("WITH RECURSIVE geometry
               (id, gmlid, gmlid_codespace, parent_id, root_id, is_solid,
        is_composite, is_triangulated, is_xlink, is_reverse, geometry, level) " +
        " AS (SELECT sg.*, 1 AS level FROM surface_geometry sg WHERE sg.id=?
               UNION ALL " +
        " SELECT sg.*, g.level + 1 AS level FROM
               surface_geometry sg, geometry g WHERE sg.parent_id=g.id)" +
        " SELECT * FROM geometry ORDER BY level asc");

98      // SURFACE_GEOMETRY_SEQ.nextval
        nextval('SURFACE_GEOMETRY_ID_SEQ')
```

B.3.4. Die „nologging“ Option

Es gibt keine `nologging` Option für `CREATE TABLE` Statements in *PostgreSQL.*

de.tub.citydb.modules.citygml.common.database.cache.model.**CacheTableModel**

```
95      // " nologging" +
```

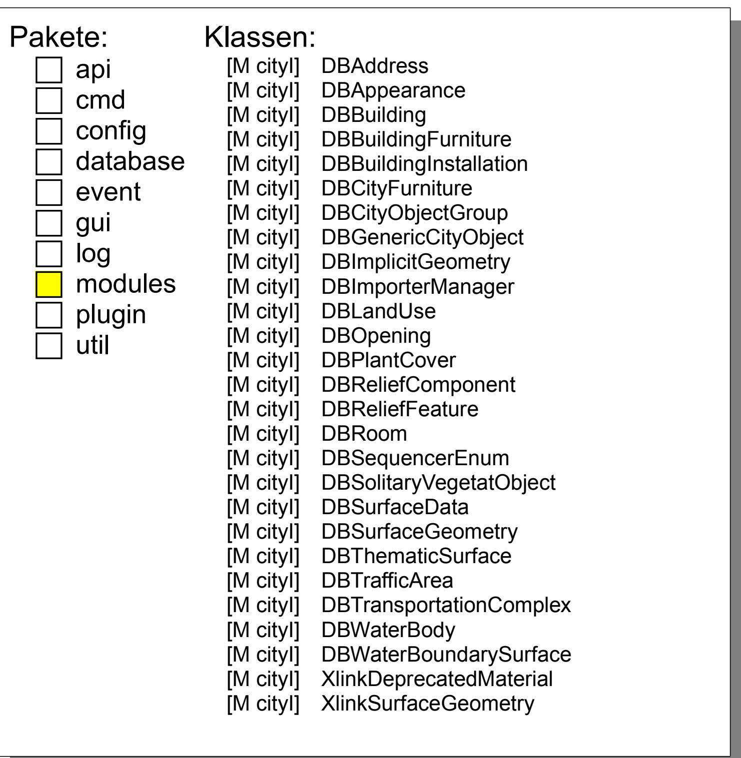
de.tub.citydb.modules.citygml.common.database.cache.**HeapCacheTable**

```
158     model.createIndexes(conn, tableName/*, "nologging"*/);
```

B.3.5. Datentypen in Cache Tabellen

Im Ordner common.database.cache.model mussten die Datentypen für temporäre Tabellen an das DBMS angepasst werden.

B.4. Implizite Sequenzen



Die ID-Spalten der Tabellen, die als Primary Keys gesetzt sind, wurden in *PostgreSQL* mit dem Datentyp SERIAL definiert. Wie in der Arbeit beschrieben, erstellt SERIAL implizit eine Sequenz, deren Namen sich aus dem Tabellen- und Spaltennamen sowie der Endung SEQ zusammensetzt. In den Skripten für die *Oracle*-Fassung wurden die Namen der Sequenzen, weil diese explizit definiert werden mussten, teilweise abgekürzt und beinhalten nicht das Kürzel „ID“ für den Spaltennamen. Diese Falle musste im Java-Code berücksichtigt werden, z.B. in der Klasse DBSequencerEnum und all ihren Aufrufe (siehe betroffenen Klassen in der Übersichtsbox). Ein anderer Fall (das Ausführen der Funktion `nextval`) ist in den Beispielen auf der vorangegangen Seite abgedeckt.

de.tub.citydb.modules.citygml.importer.database.content.**DBSequencerEnum**

```
32      //public enum DBSequencerEnum {
        //      ADDRESS_SEQ,
        //      APPEARANCE_SEQ,
        //      CITYOBJECT_SEQ,
        //      SURFACE_GEOMETRY_SEQ,
        //      IMPLICIT_GEOMETRY_SEQ,
        //      SURFACE_DATA_SEQ,
        public enum DBSequencerEnum {
                ADDRESS_ID_SEQ, APPEARANCE_ID_SEQ, CITYOBJECT_ID_SEQ,
                SURFACE_GEOMETRY_ID_SEQ, IMPLICIT_GEOMETRY_ID_SEQ,
                SURFACE_DATA_ID_SEQ,
```

B.5. Die Verarbeitung von Geometrien

Die Verarbeitung der Geometrien wurde bereits ausführlich in Kapitel 6.2.2. auf den Seiten 47 und 48 erklärt. Die aufgeführten Beispiele wiederholen sich in vielen Klassen.

B.5.1. Von CityGML zur 3DCityDB

de.tub.citydb.modules.citygml.importer.database.content.**DBAddress**

```
91      // private DBSdoGeometry sdoGeometry;
wdh+    private DBStGeometry stGeometry;

106     // sdoGeometry = (DBSdoGeometry)dbImporterManager.getDBImporter(
wdh+           DBImporterEnum.SDO_GEOMETRY);
        stGeometry = (DBStGeometry)dbImporterManager.getDBImporter(
               DBImporterEnum.ST_GEOMETRY);

133     // JGeometry multiPoint = null;
wdh+    PGgeometry multiPoint = null;

224     // multiPoint = sdoGeometry.getMultiPoint(address.getMultiPoint());
wdh+    multiPoint = stGeometry.getMultiPoint(address.getMultiPoint());

        // if (multiPoint != null) {
        //     Struct multiPointObj= SyncJGeometry.syncStore(multiPoint,batchConn);
        //     psAddress.setObject(8, multiPointObj);
```

```
// } else
//     psAddress.setNull(8, Types.STRUCT, "MDSYS.SDO_GEOMETRY");
if (multiPoint != null) {
       psAddress.setObject(8, multiPoint);
} else
       psAddress.setNull(8, Types.OTHER, "ST_GEOMETRY");
```

de.tub.citydb.modules.citygml.importer.database.content.**DBCityObject**

```
211     // double[] ordinates = new double[points.size()];
wdh+    // int i = 0;
        // for (Double point : points)
        //     ordinates[i++] = point.doubleValue();
        // JGeometry boundedBy =
        //     JGeometry.createLinearPolygon(ordinates, 3, dbSrid);
        // STRUCT obj = SyncJGeometry.syncStore(boundedBy, batchConn);
        //
        // psCityObject.setObject(4, obj);
        String geomEWKT = "SRID=" + dbSrid + ";POLYGON((";
        for (int i=0; i<points.size(); i+=3){
               geomEWKT += points.get(i) + " " + points.get(i+1) + " " +
                      points.get(i+2) + ",";
        }
        geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
        geomEWKT += "))";

        Geometry boundedBy = PGgeometry.geomFromString(geomEWKT);
        PGgeometry pgBoundedBy = new PGgeometry(boundedBy);
        psCityObject.setObject(4, pgBoundedBy);
```

de.tub.citydb.modules.citygml.importer.database.content.**DBCityObject**

```
68      // SDO_GEOMETRY();
        ST_GEOMETRY();
```

de.tub.citydb.modules.citygml.importer.database.content.**DBStGeometry**

```
88      // public JGeometry getPoint(PointProperty pointProperty) {
wdh     //     JGeometry pointGeom = null;
        public PGgeometry getPoint(PointProperty pointProperty) throws
        SQLException {
               Geometry pointGeom = null;

99      // double[] coords = new double[values.size()];
        // int i = 0;
        // for (Double value : values)
        //     coords[i++] = value.doubleValue();
        // pointGeom = JGeometry.createPoint(coords, 3, dbSrid);
        pointGeom = PGgeometry.geomFromString("SRID=" + dbSrid + ";POINT(" +
               values.get(0) + " " + values.get(1) + " " + values.get(2) + ")");

171     // if (!pointList.isEmpty()) {
wdh     //     Object[] pointArray = new Object[pointList.size()];
        //     int i = 0;
        //     for (List<Double> coordsList : pointList) {
        //            if (affineTransformation)
        //                   dbImporterManager.getAffineTransformer().
        //                          transformCoordinates(coordsList);
        //
```

```
        //     double[] coords = new double[3];
        //
        //     coords[0] = coordsList.get(0).doubleValue();
        //     coords[1] = coordsList.get(1).doubleValue();
        //     coords[2] = coordsList.get(2).doubleValue();
        //
        //     pointArray[i++] = coords;
        //     }
        //     multiPointGeom = JGeometry.createMultiPoint(pointArray, 3, dbSrid);
        // }
        // }
        // return multiPointGeom;
if (!pointList.isEmpty()) {
                String geomEWKT = "SRID=" + dbSrid + ";MULTIPOINT(";

                for (List<Double> coordsList : pointList){

                        if (affineTransformation)
                                dbImporterManager.getAffineTransformer().
                                        transformCoordinates(coordsList);
                        geomEWKT += coordsList.get(0) + " " + coordsList.get(1) + " "
                                + coordsList.get(2) + ",";
                }
                geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
                geomEWKT += ")";

                multiPointGeom = PGgeometry.geomFromString(geomEWKT);
        }
        }
        PGgeometry pgMultiPointGeom = new PGgeometry(multiPointGeom);
        return pgMultiPointGeom;

213     // if (!pointList.isEmpty()) {
wdh     //     Object[] pointArray = new Object[pointList.size()];
        //     int i = 0;
        //     for (List<Double> coordsList : pointList) {
        //             if (affineTransformation)
        //                     dbImporterManager.getAffineTransformer().
        //                             transformCoordinates(coordsList);
        //             double[] coords = new double[coordsList.size()];
        //             int j = 0;
        //             for (Double coord : coordsList)
        //                     coords[j++] = coord.doubleValue();
        //
        //             pointArray[i++] = coords;
        //     }
        //     multiCurveGeom = JGeometry.createLinearMultiLineString(pointArray,
        //     3, dbSrid);
        // }
        if (!pointList.isEmpty()) {
                String geomEWKT = "SRID=" + dbSrid + ";MULTILINESTRING((";

                for (List<Double> coordsList : pointList) {
                        if (affineTransformation)
                                dbImporterManager.getAffineTransformer().
                                        transformCoordinates(coordsList);

                        for (int i=0; i<coordsList.size(); i+=3){
                                geomEWKT += coordsList.get(i) + " " +
                                coordsList.get(i+1) + " " + coordsList.get(i+2) + ",";
                        }
                        geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
                        geomEWKT += "),(";
                }
```

```
geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 2);
geomEWKT += ")";
multiCurveGeom = PGgeometry.geomFromString(geomEWKT);
}
```

de.tub.citydb.modules.citygml.importer.database.content.**DBSurfaceData**

```
437   // JGeometry geom = new JGeometry(coords.get(0), coords.get(1), dbSrid);
      // STRUCT obj = SyncJGeometry.syncStore(geom, batchConn);
      // psSurfaceData.setObject(15, obj);
      Geometry geom = PGgeometry.geomFromString("SRID=" + dbSrid + ";POINT(" +
            coords.get(0) + " " + coords.get(1) + ")");
      PGgeometry pgGeom = new PGgeometry(geom);
      psSurfaceData.setObject(15, pgGeom);
```

de.tub.citydb.modules.citygml.importer.database.xlink.resolver.**XlinkSurfaceGeometry**

```
283   // if (reverse) {
      //     int[] elemInfoArray = geomNode.geometry.getElemInfo();
      //     double[] ordinatesArray = geomNode.geometry.getOrdinatesArray();
      //
      //     if (elemInfoArray.length < 3 || ordinatesArray.length == 0) {
      //         geomNode.geometry = null;
      //         return;
      //     }
      //
      //     // we are pragmatic here. if elemInfoArray contains more than one
      //     // entry, we suppose we have one outer ring and anything else are
      //     // inner rings.
      //     List<Integer> ringLimits = new ArrayList<Integer>();
      //     for (int i = 3; i < elemInfoArray.length; i += 3)
      //         ringLimits.add(elemInfoArray[i] - 1);
      //
      //     ringLimits.add(ordinatesArray.length);
      //
      //     // ok, reverse polygon according to this info
      //     Object[] pointArray = new Object[ringLimits.size()];
      //     int ringElem = 0;
      //     int arrayIndex = 0;
      //     for (Integer ringLimit : ringLimits) {
      //         double[] coords = new double[ringLimit - ringElem];
      //
      //         for (int i=0, j=ringLimit-3; j>=ringElem; j-=3, i+=3) {
      //             coords[i] = ordinatesArray[j];
      //             coords[i + 1] = ordinatesArray[j + 1];
      //             coords[i + 2] = ordinatesArray[j + 2];
      //         }
      //
      //         pointArray[arrayIndex++] = coords;
      //         ringElem = ringLimit;
      //     }
      //
      //     JGeometry geom = JGeometry.createLinearPolygon(PointArray,
      //         geomNode.geometry.getDimensions(),
      //             geomNode.geometry.getSrid());
      //
      //     geomNode.geometry = geom;
      // }
```

```
        if (reverse) {
                String geomEWKT = "SRID=" + geomNode.geometry.getSrid() +
                        ";POLYGON((";
                Polygon polyGeom = (Polygon) geomNode.geometry;
                int dimensions = geomNode.geometry.getDimension();

                for (int i = 0; i < polyGeom.numRings(); i++){
                if (dimensions == 2)
                        for (int j=0; j<polyGeom.getRing(i).numPoints(); j++){
                                geomEWKT += polyGeom.getRing(i).getPoint(j).x + " " +
                        polyGeom.getRing (i).getPoint(j).y + ",";
                        }

                if (dimensions == 3)
                        for (int j=0; j<polyGeom.getRing (i).numPoints(); j++){
                                geomEWKT += polyGeom.getRing (i).getPoint(j).x + " " +
                        polyGeom.getRing(i).getPoint(j).y + " " +
                        polyGeom.getRing(i).getPoint(j).z + ",";
                        }

                        geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
                        geomEWKT += "),(";
                }


                geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 2);
                geomEWKT += ")";

                Geometry geom = PGgeometry.geomFromString(geomEWKT);
                geomNode.geometry = geom;
        }

382     // protected JGeometry geometry;
wdh+    protected Geometry geometry;
```

de.tub.citydb.modules.citygml.importer.database.xlink.resolver.**XlinkWorldFile**

```
134     // JGeometry geom = new JGeometry(content.get(4), content.get(5), dbSrid);
        // STRUCT obj = JGeometry.store(geom, batchConn);
        Point ptGeom = new Point(content.get(4), content.get(5));
        Geometry geom = PGgeometry.geomFromString(
                "SRID=" + dbSrid + ";" + ptGeom);
        PGgeometry pgGeom = new PGgeometry(geom);
```

B.5.2. Von der 3DCityDB zu CityGML

de.tub.citydb.modules.citygml.exporter.database.content.**DBAppearance**

```
822     // STRUCT struct = (STRUCT)rs.getObject("GT_REFERENCE_POINT");
wdh+    // if (!rs.wasNull() && struct != null) {
        //      JGeometry jGeom = JGeometry.load(struct);
        //      double[] point = jGeom.getPoint();
        //      if (point != null && point.length >= 2) {
        //              Point referencePoint = new PointImpl();
        //              List<Double> value = new ArrayList<Double>();
        //              value.add(point[0]);
        //              value.add(point[1]);
```

```
        PGgeometry pgGeom = (PGgeometry)rs.getObject("GT_REFERENCE_POINT");
        if (!rs.wasNull() && pgGeom != null) {
                Geometry geom = pgGeom.getGeometry();
                Point referencePoint = new PointImpl();
                List<Double> value = new ArrayList<Double>();
                value.add(geom.getPoint(0).getX());
                value.add(geom.getPoint(0).getY());
```

de.tub.citydb.modules.citygml.exporter.database.content.**DBCityObject**

```
164     // double[] points = geom.getMBR();

170     // if (geom.getDimension() == 2) {
        //      lower = new Point(points[0], points[1], 0);
        //      upper = new Point(points[2], points[3], 0);
        // } else {
        //      lower = new Point(points[0], points[1], points[2]);
        //      upper = new Point(points[3], points[4], points[5]);
        if (geom.getDimension() == 2) {
                lower = new Point(geom.getFirstPoint().x, geom.getFirstPoint().y,0);
                upper = new Point(geom.getPoint(2).x, geom.getPoint(2).y, 0);
        } else {
                lower = new Point(geom.getFirstPoint().x, geom.getFirstPoint().y,
                        geom.getFirstPoint().z);
                upper = new Point(geom.getPoint(2).x, geom.getPoint(2).y,
                        geom.getPoint(2).z);
```

de.tub.citydb.modules.citygml.exporter.database.content.**DBGeneralization**

```
121     // double[] points = geom.getOrdinatesArray();
        // Point lower = new Point(points[0], points[1], points[2]);
        // Point upper = new Point(points[3], points[4], points[5]);
        Point lower = new Point(geom.getFirstPoint().x, geom.getFirstPoint().y,
        geom.getFirstPoint().z);
        Point upper = new Point(geom.getPoint(2).x, geom.getPoint(2).y,
                geom.getPoint(2).z);
```

de.tub.citydb.modules.citygml.exporter.database.content.**DBStGeometry**

```
94      // public PointProperty getPoint(JGeometry geom, boolean setSrsName) {
        //      PointProperty pointProperty = null;
        //      if (geom != null && geom.getType() == JGeometry.GTYPE_POINT) {
        //              pointProperty = new PointPropertyImpl();
        //              int dimensions = geom.getDimensions();
        //
        //              double[] pointCoord = geom.getPoint();
        //
        //              if (pointCoord != null && pointCoord.length >= dimensions) {
        //                      Point point = new PointImpl();
        //
        //                      List<Double> value = new ArrayList<Double>();
        //                      for (int i = 0; i < dimensions; i++)
        //                              value.add(pointCoord[i]);
        public PointProperty getPoint(Geometry geom, boolean setSrsName) {
                PointProperty pointProperty = null;

                if (geom != null && geom.getType() == 1) {
                        pointProperty = new PointPropertyImpl();
                        int dimensions = geom.getDimension();
```

```
                if (dimensions == 2) {
                        Point point = new PointImpl();
                        List<Double> value = new ArrayList<Double>();
                        value.add(geom.getPoint(0).getX());
                        value.add(geom.getPoint(0).getY());
                        .
                        .
                if (dimensions == 3) {
                        Point point = new PointImpl();
                        List<Double> value = new ArrayList<Double>();
                        value.add(geom.getPoint(0).getX());
                        value.add(geom.getPoint(0).getY());
                        value.add(geom.getPoint(0).getZ());

140   // public PolygonProperty getPolygon(JGeometry geom, boolean setSrsName) {
      //      PolygonProperty polygonProperty = null;
      //
      //      if (geom != null && geom.getType() == JGeometry.GTYPE_POLYGON) {
      //              polygonProperty = new PolygonPropertyImpl();
      //              Polygon polygon = new PolygonImpl();
      //              int dimensions = geom.getDimensions();
      //              int[] elemInfoArray = geom.getElemInfo();
      //              double[] ordinatesArray = geom.getOrdinatesArray();
      //
      //              if (elemInfoArray.length < 3 || ordinatesArray.length == 0)
      //                      return null;
      //
      //              List<Integer> ringLimits = new ArrayList<Integer>();
      //              for (int i = 3; i < elemInfoArray.length; i += 3)
      //                      ringLimits.add(elemInfoArray[i] - 1);
      //
      //              ringLimits.add(ordinatesArray.length);
      //              boolean isExterior = elemInfoArray[1] == 1003;
      //              int ringElem = 0;
      //              for (Integer curveLimit : ringLimits) {
      //                      List<Double> values = new ArrayList<Double>();
      //
      //                      for ( ; ringElem < curveLimit; ringElem++)
      //                              values.add(ordinatesArray[ringElem]);
      //
      //                      if (isExterior) {

      public PolygonProperty getPolygon(Geometry geom, boolean setSrsName) {

      PolygonProperty polygonProperty = null;

      if (geom != null && geom.getType() == 3) {
              polygonProperty = new PolygonPropertyImpl();
              Polygon polygon = new PolygonImpl();
              int dimensions = geom.getDimension();

              if (geom.getValue() == null)
                      return null;

              org.postgis.Polygon polyGeom = (org.postgis.Polygon) geom;

              for (int i = 0; i < polyGeom.numRings(); i++){
                      List<Double> values = new ArrayList<Double>();

              if (dimensions == 2)
              for (int j=0; j<polyGeom.getRing(i).numPoints(); j++){
              values.add(polyGeom.getRing(i).getPoint(j).x);
                      values.add(polyGeom.getRing(i).getPoint(j).y);
              }
```

```
if (dimensions == 3)
        for (int j=0; j<polyGeom.getRing (i).numPoints(); j++){
                values.add(polyGeom.getRing (i).getPoint(j).x);
                values.add(polyGeom.getRing (i).getPoint(j).y);
                values.add(polyGeom.getRing (i).getPoint(j).z);
        }
        //isExterior
        if (i == 0) {
```

208
wdh

```
// public MultiPointProperty getMultiPointProperty(JGeometry geom, boolean
// setSrsName) {
//      MultiPointProperty multiPointProperty = null;
//
//      if (geom != null) {
//              multiPointProperty = new MultiPointPropertyImpl();
//              MultiPoint multiPoint = new MultiPointImpl();
//              int dimensions = geom.getDimensions();
//
//      if (geom.getType() == JGeometry.GTYPE_MULTIPOINT) {
//              double[] ordinates = geom.getOrdinatesArray();
//
//              for (int i = 0; i < ordinates.length; i += dimensions) {
//                      Point point = new PointImpl();
//
//              List<Double> value = new ArrayList<Double>();
//
//              for (int j = 0; j < dimensions; j++)
//                      value.add(ordinates[i + j]);
//                      ...                     .
//              }
//      } else if (geom.getType() == JGeometry.GTYPE_POINT) {
//              ...
public MultiPointProperty getMultiPointProperty(Geometry geom, boolean
setSrsName) {
MultiPointProperty multiPointProperty = null;

if (geom != null) {
        multiPointProperty = new MultiPointPropertyImpl();
        MultiPoint multiPoint = new MultiPointImpl();
        int dimensions = geom.getDimension();

if (geom.getType() == 4) {
        List<Double> value = new ArrayList<Double>();
        Point point = new PointImpl();

        if (dimensions == 2)
                for (int i = 0; i < geom.numPoints(); i++) {
                        value.add(geom.getPoint(i).x);
                        value.add(geom.getPoint(i).y);
                }
        if (dimensions == 3)
                for (int i = 0; i < geom.numPoints(); i++) {
                        value.add(geom.getPoint(i).x);
                        value.add(geom.getPoint(i).y);
                        value.add(geom.getPoint(i).z);
                }
        ...
        }
}
else if (geom.getType() == 1) {
        Point point = new PointImpl();
        List<Double> value = new ArrayList<Double>();
        value.add(geom.getPoint(0).x);
        value.add(geom.getPoint(0).y);
```

```
if (dimensions == 3)
        value.add(geom.getPoint(0).z);

355  // public MultiCurveProperty getMultiCurveProperty(JGeometry geom, boolean
wdh  // setSrsName) {
     //     MultiCurveProperty multiCurveProperty = null;
     //
     //     if (geom != null) {
     //             multiCurveProperty = new MultiCurvePropertyImpl();
     //             MultiCurve multiCurve = new MultiCurveImpl();
     //             int dimensions = geom.getDimensions();
     //
     //             if (geom.getType() == JGeometry.GTYPE_MULTICURVE ) {
     //                     int[] elemInfoArray = geom.getElemInfo();
     //                     double[] ordinatesArray = geom.getOrdinatesArray();
     //
     //                     if (elemInfoArray.length < 3 ||
     //                             ordinatesArray.length == 0)
     //                                     return null;
     //                     List<Integer> curveLimits = new ArrayList<Integer>();
     //                             for (int i = 3; i < elemInfoArray.length; i += 3)
     //                                     curveLimits.add(elemInfoArray[i] - 1);
     //
     //                     curveLimits.add(ordinatesArray.length);
     //
     //                     int curveElem = 0;
     //                     for (Integer curveLimit : curveLimits) {
     //                             List<Double> values = new ArrayList<Double>();
     //                             for ( ; curveElem < curveLimit; curveElem++)
     //                                     values.add(ordinatesArray[curveElem]);
     //                             ...
     //                             curveElem = curveLimit;
     //                     }
     //             }
     //             else if (geom.getType() == JGeometry.GTYPE_CURVE ) {
     //                     double[] ordinatesArray = geom.getOrdinatesArray();
     //                     List<Double> value = new ArrayList<Double>();
     //                     for (int i = 0; i < ordinatesArray.length; i++)
     //                             value.add(ordinatesArray[i]);
     public MultiCurveProperty getMultiCurveProperty(Geometry geom, boolean
     setSrsName) {
             MultiCurveProperty multiCurveProperty = null;

             if (geom != null) {
             multiCurveProperty = new MultiCurvePropertyImpl();
             MultiCurve multiCurve = new MultiCurveImpl();
             int dimensions = geom.getDimension();

             if (geom.getType() == 5) {
                     MultiLineString mlineGeom = (MultiLineString)geom;

                     for (int i = 0; i < mlineGeom.numLines(); i++){

                             List<Double> values = new ArrayList<Double>();

                             if (dimensions == 2)
                                     for (int j=0; j<mlineGeom.getLine(i).numPoints();
                                     j++){
                                       values.add(mlineGeom.getLine(i).getPoint(j).x);
                                       values.add(mlineGeom.getLine(i).getPoint(j).y);
                                     }
                             if (dimensions == 3)
                                     for (int j=0; j<mlineGeom.getLine(i).numPoints();
                                     j++){
```

```
                        values.add(mlineGeom.getLine(i).getPoint(j).x);
                        values.add(mlineGeom.getLine(i).getPoint(j).y);
                        values.add(mlineGeom.getLine(i).getPoint(j).z);
                    }
                    ...
                }
            }
            else if (geom.getType() == 2) {
                List<Double> value = new ArrayList<Double>();

                if (dimensions == 2)
                    for (int i = 0; i < geom.numPoints(); i++){
                        value.add(geom.getPoint(i).x);
                        value.add(geom.getPoint(i).y);
                    }
                if (dimensions == 3)
                    for (int i = 0; i < geom.numPoints(); i++){
                        value.add(geom.getPoint(i).x);
                        value.add(geom.getPoint(i).y);
                        value.add(geom.getPoint(i).z);
                    }
```

de.tub.citydb.util.database.**DBUtil**

```
308    // STRUCT struct = (STRUCT)rs.getObject(1);
wdh+   // if (!rs.wasNull() && struct != null) {
       //     JGeometry jGeom = JGeometry.load(struct);
       //     int dim = jGeom.getDimensions();
       PGgeometry pgGeom = (PGgeometry)rs.getObject(1);
       if (!rs.wasNull() && pgGeom != null) {
           Geometry geom = pgGeom.getGeometry();
           int dim = geom.getDimension();
```

B.6. Die Verarbeitung von Texturen und binären Daten

Pakete:
- [ ] api
- [ ] cmd
- [ ] config
- [ ] database
- [ ] event
- [ ] gui
- [ ] log
- [x] modules
- [ ] plugin
- [ ] util

Klassen:
[M cityE] DBAppearance
[M cityE] DBXlinkExporterLibraryObject
[M cityE] DBXlinkExporterTextureImage
[M cityI] XlinkLibraryObject
[M cityI] XlinkTextureImage

Die Unterschiede bei der Verarbeitung von Texturen zwischen *Oracle* und *PostgreSQL* sind ebenfalls in Kapitel 6.2.2. besprochen worden. Auf Datenbank-Seite beschränkte sich die Verwendung von ORDImage-spezifischen Methoden auf die Klasse `DBAppearance`. Die Funktion `ORDImage.getMimeType` konnte nicht übersetzt werden. Die Information aus der Spalte `sd.TEX_MIME_TYPE` muss an dieser Stelle genügen.

de.tub.citydb.modules.citygml.exporter.database.content.**DBAppearance**
```
138    // nvl(sd.TEX_IMAGE.getContentLength(), 0) as DB_TEX_IMAGE_SIZE,
wdh    // sd.TEX_IMAGE.getMimeType() as DB_TEX_IMAGE_MIME_TYPE, sd.TEX_MIME_TYPE,
       COALESCE(length(sd.TEX_IMAGE), 0) as DB_TEX_IMAGE_SIZE, sd.TEX_MIME_TYPE,
```

Der Datenaustausch mit Spalten vom Typ BYTEA betrifft neben Texturen auch Library Objects aus der Tabelle `Implicit_Geometry`. Die Implementation in Java ist jeweils identisch. Da in *Oracle* der Datentyp BLOB verwendet wird, ergeben sich Unterschiede, die in B.5.1. veranschaulicht werden.

Neben einer Im- und Export-Logik für BYTEAs wurde auch ein alternativer Weg mit Large Objects (LOB) programmiert. Der Datentyp BYTEA hat den Nachteil, dass er nicht sequentiell in ein Stream-Objekt gelesen werden kann und die komplette Datei in den Hauptspeicher geladen werden muss. Häufig sind die zu einem CityGML-Datensatz referenzierten Texturdateien sehr klein, doch es existieren auch Beispiele, die Texturatlanten verwenden. Bei einem Texturatlas werden viele Texturen in einer Datei gesammelt, und über Texturkoordinaten ihren Flächen zugewiesen. Das ist sinnvoll, wenn gleiche Texturen für mehrere Objekte (z.B. Fenster) oder Gebäude benutzt werden. Wenn für jede Fläche ein Texturatlas in den Hauptspeicher gelesen werden muss und dieser einige MB groß ist, dann dauert es bedingt durch das Multithreading nicht lange bis der Java Heap Space überläuft. LOBs hätten diesen Nachteil nicht, da sie streambasiert verarbeitet werden können.

In den getesteten Fällen ergab sich bisher keine Verkürzung der Ausführungszeit. LOBs werden separat im *PostgreSQL* Datei-System abgelegt und in der jeweiligen Tabellenspalte über einen „Object Identifier" Datentyp (OID) referenziert. Bei einem BYTEA befindet sich hingegen die komplette Datei in der Tabelle. Dementsprechend langsam wäre bei vielen Einträgen das Ausrufen des kompletten Tabelleninhaltes. Wenn Einträge aus einer Spalte vom Datentyp OID gelöscht werden, wird nur die Referenz nicht aber die Datei entfernt. Dafür existieren bestimmte Funktionen (`vacuumlo`, `lo_unlink`). Um dem Benutzer die etwas komplizierte Handhabung von LOBs zu ersparen, wird in der Release-Version die Speicherung in BYTEA bevorzugt.

B.6.1. Der Import von Texturen und binären Daten

de.tub.citydb.modules.citygml.importer.database.xlink.resolver.**XlinkTextureImage**

```
77      // psPrepare = externalFileConn.prepareStatement(
              "update SURFACE_DATA set TEX_IMAGE=ordimage.init() where ID=?");
        // psSelect = externalFileConn.prepareStatement(
              "select TEX_IMAGE from SURFACE_DATA where ID=? for update");
        // psInsert = (OraclePreparedStatement)externalFileConn.prepareStatement(
              "update SURFACE_DATA set TEX_IMAGE=? where ID=?");
        psInsert = externalFileConn.prepareStatement(
              "update SURFACE_DATA set TEX_IMAGE=? where ID=?");

113+    // // second step: prepare ORDIMAGE
        // psPrepare.setLong(1, xlink.getId());
        // psPrepare.executeUpdate();
        //
        // // third step: get prepared ORDIMAGE to fill it with contents
        // psSelect.setLong(1, xlink.getId());
        // OracleResultSet rs = (OracleResultSet)psSelect.executeQuery();
        //     if (!rs.next()) {
        //           LOG.error("Database error while importing texture file '" +
        //                 imageFileName + "'.");
        //
        //           rs.close();

        //           externalFileConn.rollback();
        //           return false;
        //     }

120     // OrdImage imgProxy = (OrdImage)rs.getORAData(
        //     1,OrdImage.getORADataFactory());
        // rs.close();
        // boolean letDBdetermineProperties = true;
        // if (isRemote) {
        //     InputStream stream = imageURL.openStream();
        //     imgProxy.loadDataFromInputStream(stream);
        // } else {
        //     imgProxy.loadDataFromFile(imageFileName);
        //
        //     // determing image formats by file extension
        //     int index = imageFileName.lastIndexOf('.');
        //     if (index != -1) {
        //           String extension = imageFileName.substring(
        //                 index + 1, imageFileName.length());
        //
```

```
        //          if (extension.toUpperCase().equals("RGB")) {
        //                  imgProxy.setMimeType("image/rgb");
        //                  imgProxy.setFormat("RGB");
        //                  imgProxy.setContentLength(1);
        //
        //                  letDBdetermineProperties = false;
        //          }
        //      }
        // }
        // if (letDBdetermineProperties)
        //      imgProxy.setProperties();
        //
        // psInsert.setORAData(1, imgProxy);
        // psInsert.setLong(2, xlink.getId());
        // psInsert.execute();
        // imgProxy.close();
        InputStream in = null;
        if (isRemote) {
                in = imageURL.openStream();
        } else {
                in = new FileInputStream(imageFile);
        }

/*      // insert large object (OID) data type into database

        // All LargeObject API calls must be within a transaction block
        externalFileConn.setAutoCommit(false);

        // Get the Large Object Manager to perform operations with
        LargeObjectManager lobj =
                ((org.postgresql.PGConnection)externalFileConn).getLargeObjectAPI();

        // Create a new large object
        long oid = lobj.createLO(LargeObjectManager.READ | LargeObjectManager.WRITE);

        // Open the large object for writing
        LargeObject obj = lobj.open(oid, LargeObjectManager.WRITE);

        // Copy the data from the file to the large object
        byte buf[] = new byte[2048];
        int s, tl = 0;

        while ((s = in.read(buf, 0, 2048)) > 0) {
                obj.write(buf, 0, s);
                tl = tl + s;
        }

        // Close the large object
        obj.close();
*/
//      psInsert.setLong(1, oid); // for large object
        psInsert.setBinaryStream(1, in, in.available()); // for bytea
        psInsert.setLong(2, xlink.getId());
        psInsert.execute();

        in.close()
        externalFileConn.commit();
        return true;
```

de.tub.citydb.modules.citygml.importer.database.xlink.resolver.**XlinkLibraryObject**

```
74      // psPrepare = externalFileConn.prepareStatement(
        //   "update IMPLICIT_GEOMETRY set LIBRARY_OBJECT=empty_blob() where ID=?");
        // psSelect = externalFileConn.prepareStatement(
        //   "select LIBRARY_OBJECT from IMPLICIT_GEOMETRY where ID=? for update");
        psInsert = externalFileConn.prepareStatement(
          "update IMPLICIT_GEOMETRY set LIBRARY_OBJECT=? where ID=?");

80+     // // first step: prepare BLOB
        // psPrepare.setLong(1, xlink.getId());
        // psPrepare.executeUpdate();
        //
        // // second step: get prepared BLOB to fill it with contents
        // psSelect.setLong(1, xlink.getId());
        // OracleResultSet rs = (OracleResultSet)psSelect.executeQuery();
        // if (!rs.next()) {
        //     LOG.error("Database error while importing library object: " +
                       objectFileName);
        //
        //     rs.close();
        //     externalFileConn.rollback();
        //     return false;
        // }
        //
        // BLOB blob = rs.getBLOB(1);
        // rs.close();

126+    // OutputStream out = blob.setBinaryStream(1L);
        //
        // int size = blob.getBufferSize();
        // byte[] buffer = new byte[size];
        // int length = -1;
        //
        // while ((length = in.read(buffer)) != -1)
        //     out.write(buffer, 0, length);
        //
        // in.close();
        // blob.close();
        // out.close();
        // externalFileConn.commit();
        // return true;

/*      // insert large object (OID) data type into database

        // All LargeObject API calls must be within a transaction block
        externalFileConn.setAutoCommit(false);

        // Get the Large Object Manager to perform operations with
        LargeObjectManager lobj =
               ((org.postgresql.PGConnection)externalFileConn).getLargeObjectAPI();

        // Create a new large object
        long oid = lobj.createLO(LargeObjectManager.READ | LargeObjectManager.WRITE);

        // Open the large object for writing
        LargeObject obj = lobj.open(oid, LargeObjectManager.WRITE);

        // Copy the data from the file to the large object
        byte buf[] = new byte[2048];
        int s, tl = 0;
```

```
        while ((s = in.read(buf, 0, 2048)) > 0) {
              obj.write(buf, 0, s);
              tl = tl + s;
        }

        // Close the large object
        obj.close();
*/

        // insert bytea data type into database
        // psInsert.setLong(1, oid); // for large object
        psInsert.setBinaryStream(1, in, in.available()); // for bytea
        psInsert.setLong(2, xlink.getId());
        psInsert.execute();

        in.close();
        externalFileConn.commit();
        return true;
```

B.6.1. Der Export von Texturen und binären Daten

de.tub.citydb.modules.citygml.exporter.database.xlink.**DBXlinkExporterTextureImage**
de.tub.citydb.modules.citygml.exporter.database.xlink.**DBXlinkExporterLibraryObject**

```
128     // OracleResultSet rs = (OracleResultSet)psTextureImage.executeQuery();
wdh+    ResultSet rs = (ResultSet)psTextureImage.executeQuery();

143     // // read oracle image data type
wdh+    // OrdImage imgProxy = (OrdImage)rs.getORAData(
        //     1, OrdImage.getORADataFactory());
        // rs.close();
        //
        // if (imgProxy == null) {
        //     LOG.error("Database error while reading texture file: " + fileName);
        //     return false;
        // }
        //
        // try {
        //     imgProxy.getDataInFile(fileURI);
        // } catch (IOException ioEx) {
        //     LOG.error("Failed to write texture file " + fileName + ": " +
        //             ioEx.getMessage());
        //     return false;
        // } finally {
        //     imgProxy.close();
        // }
```

Verwendete Methode:

```
        byte[] imgBytes = rs.getBytes(1);
        try {
              FileOutputStream fos = new FileOutputStream(fileURI);
              fos.write(imgBytes);
              fos.close();
        } catch (FileNotFoundException fnfEx) {
              LOG.error("File not found " + fileName + ": " + fnfEx.getMessage());
        } catch (IOException ioEx) {
        LOG.error("Failed to write texture file " + fileName + ": " +
              ioEx.getMessage());
        return false;
        }
```

Alternativer Weg:

```
        InputStream imageStream = rs.getBinaryStream(1);
        if (imageStream == null) {
                LOG.error("Database error while reading texture file: " + fileName);
                return false;
        }
        try {
                byte[] imgBuffer = new byte[1024];
                FileOutputStream fos = new FileOutputStream(fileURI);
                int l;
                while ((l = imageStream.read(imgBuffer)) > 0) {
                    fos.write(imgBuffer, 0, l);
              }
                fos.close();
        } catch (FileNotFoundException fnfEx) {
                LOG.error("File not found " + fileName + ": " + fnfEx.getMessage());
        } catch (IOException ioEx) {
                LOG.error("Failed to write texture file " + fileName + ": " +
                        ioEx.getMessage());
                return false; }
```

Large Objects Methode:

```
        // Get the Large Object Manager to perform operations with
        LargeObjectManager lobj =
                ((org.postgresql.PGConnection)connection).getLargeObjectAPI();

        // Open the large object for reading
        long oid = rs.getLong(1);
        if (oid == 0) {
                LOG.error("Database error while reading library object: " + fileName);
                        return false;
        }
        LargeObject obj = lobj.open(oid, LargeObjectManager.READ);

        // Read the data
        byte buf[] = new byte[obj.size()];
        obj.read(buf, 0, obj.size());

        // Write the data
        try {
                FileOutputStream fos = new FileOutputStream(fileURI);
                fos.write(buf, 0, obj.size());
                obj.close();
                fos.close();
        } catch (FileNotFoundException fnfEx) {
                LOG.error("File not found " + fileName + ": " + fnfEx.getMessage());
        } catch (IOException ioEx) {
                LOG.error("Failed to write texture file " + fileName + ": " +
                        ioEx.getMessage());
                return false;
                }

        connection.commit();
```

B.7. Die Batchsize von PostgreSQL

Pakete:

- [ ] api
- [ ] cmd
- [x] config
- [ ] database
- [ ] event
- [ ] gui
- [ ] log
- [x] modules
- [ ] plugin
- [ ] util

Klassen:

[C]	Internal
[C]	UpdateBatching
[M cityE]	DBExportCache
[M cityI]	DBImportXlinkResolverWorker
[M cityI]	DBImportXlinkWorker
[M cityI]	DBAddress
[M cityI]	DBAddressToBuilding
[M cityI]	DBAppearance
[M cityI]	DBAppearToSurfaceData
[M cityI]	DBBuilding
[M cityI]	DBBuildingFurniture
[M cityI]	DBBuildingInstallation
[M cityI]	DBCityFurniture
[M cityI]	DBCityObject
[M cityI]	DBCityObjectGenericCityObject
[M cityI]	DBCityObjectGroup
[M cityI]	DBExternalReference
[M cityI]	DBGenericCityObject
[M cityI]	DBImplicitGeometry
[M cityI]	DBLandUse
[M cityI]	DBOpening
[M cityI]	DBOpeningToThemSurface
[M cityI]	DBPlantCover
[M cityI]	DBReliefComponent
[M cityI]	DBReliefFeatToRelComp
[M cityI]	DBReliefFeature
[M cityI]	DBRoom
[M cityI]	DBSolitaryVegetatObject
[M cityI]	DBSurfaceData
[M cityI]	DBSurfaceGeometry
[M cityI]	DBThematicSurface
[M cityI]	DBTrafficArea
[M cityI]	DBTransportationComplex
[M cityI]	DBWaterBodyToWaterBndSrf
[M cityI]	DBWaterBody
[M cityI]	DBWaterBoundarySurface
[M cityI]	DBImportCache
[M cityI]	DBXlinkImporterBasic
[M cityI]	DBXlinkImporterDeprecatedMaterial
[M cityI]	DBXlinkImporterGroupToCityObject
[M cityI]	DBXlinkImporterLibraryObject
[M cityI]	DBXlinkImporterLinearRing
[M cityI]	DBXlinkImporterSurfacegeometry
[M cityI]	DBXlinkImporterTextureAssociation
[M cityI]	DBXlinkImporterTextureFile
[M cityI]	DBXlinkImporterTextureParam
[M cityI]	XlinkBasic
[M cityI]	XlinkDeprecatedMaterial
[M cityI]	XlinkGroupToCityObject
[M cityI]	XlinkSurfaceGeometry
[M cityI]	XlinkTexCoordList
[M cityI]	XlinkTextureAssociation
[M cityI]	XlinkTextureParam
[M cityI]	XlinkWorldFile
[M cityI]	ResourcesPanel

Die maximale Batchsize von *PostgreSQL* wurde in der Internal-Klasse bisher nur pauschal auf 10000 gesetzt. Evtl. ist mehr möglich. In allen in der Box aufgeführten Klassen musste meist nur an einer Stelle ein Attributname von ORACLE auf *POSTGRESQL* verändert werden.

de.tub.citydb.config.internal.**Internal**

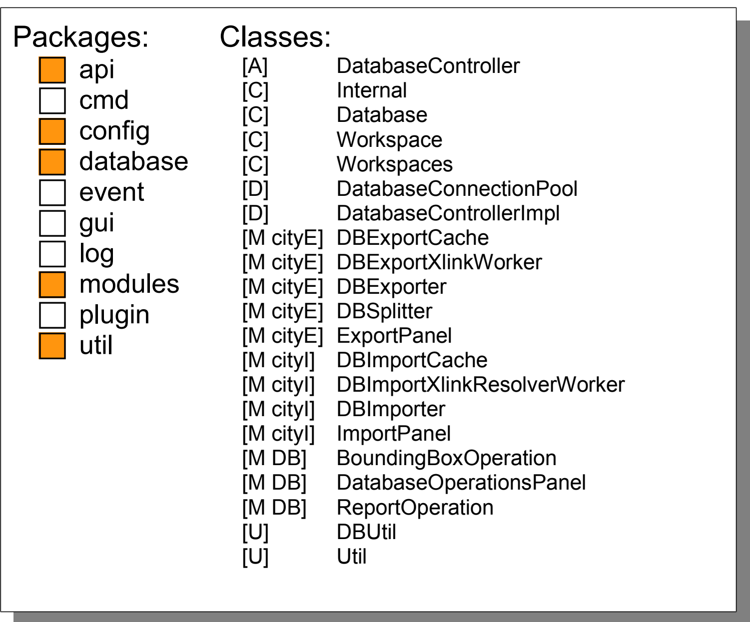
```
40    //    public static final int ORACLE_MAX_BATCH_SIZE = 65535;
      public static final int POSTGRESQL_MAX_BATCH_SIZE = 10000;
```

In den folgenden zwei Klassen konnte keine äquivalente Methode für das `PreparedStatement` gefunden werden. Der Statement-Batch wird mit `executeBatch()` ausgeführt (statt `sendBatch()`).

de.tub.citydb.modules.citygml.exporter.database.gmlid.**DBExportCache**
de.tub.citydb.modules.citygml.importer.database.gmlid.**DBImportCache**

```
84    // ((OraclePreparedStatement)psDrains[i]).setExecuteBatch(batchSize);

145   // ((OraclePreparedStatement)psDrain).sendBatch();
      psDrain.executeBatch();
```

B.8. Workspace Management



Alle Variablen und Quellcode Passagen, die sich auf das Workspace Management in *Oracle* bezogen, konnten nicht übersetzt werden und wurden auskommentiert. Aus der Übersichtsbox kann entnommen werden, welche Klassen bei einer zukünftigen Implementierung betroffen wären.

B.9. KML-Exporter

Dank des modularen Aufbaus des *Importer/Exporter* mussten bei der Portierung des *KML-Exporter* nur Klassen des kml-Moduls angepasst werden. Das Code-Design unterscheidet sich merkbar vom CityGML Im- und Export. Alle Datenbank-Abfragen sind in einer Klasse gebündelt und werden als String Konstanten an die jeweilige Klasse übergeben. Die Datenbankgeometrien werden in Arrays aufgelöst, um die KML Primitive erstellen zu können. Momentan ist es nur möglich Gebäude zu exportieren, eine sukzessive Ausweitung auf die anderen thematischen Module von CityGML ist aber angedacht.

B.9.1. Datenbank-Abfragen

de.tub.citydb.modules.kml.database.**Queries**

```
53     // public static final String GET_GMLIDS =
wdh   //      "SELECT co.gmlid, co.class_id " +
       //      "FROM CITYOBJECT co " +
       //      "WHERE " +
       //        "(SDO_RELATE(co.envelope, MDSYS.SDO_GEOMETRY(2002, ?, null, " +
       //             "MDSYS.SDO_ELEM_INFO_ARRAY(1,2,1), " +
       //             "MDSYS.SDO_ORDINATE_ARRAY(?,?,?,?,?,?)), " +
       //             "'mask=overlapbdydisjoint') ='TRUE') " +
       //      "UNION ALL " +
       //      "SELECT co.gmlid, co.class_id " +
       //      "FROM CITYOBJECT co " +
       //      "WHERE " +
       //        "(SDO_RELATE(co.envelope, MDSYS.SDO_GEOMETRY(2003, ?, null,
       //             "MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3), " +
       //             "MDSYS.SDO_ORDINATE_ARRAY(?,?,?,?)), " +
       //             "'mask=inside+coveredby') ='TRUE') " +
       //      "UNION ALL " +
       //      "SELECT co.gmlid, co.class_id " +
       //      "FROM CITYOBJECT co WHERE " +
       //        "(SDO_RELATE(co.envelope, MDSYS.SDO_GEOMETRY(2003, ?, null, " +
       //             "MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3), " +
       //             "MDSYS.SDO_ORDINATE_ARRAY(?,?,?,?)), 'mask=equal') ='TRUE') "
       //      + "ORDER BY 2"; // ORDER BY co.class_id
```

```
        public static final String GET_GMLIDS =
                "SELECT co.gmlid, co.class_id " +
                "FROM CITYOBJECT co " +
                "WHERE " +
                  // overlap
                  "ST_Relate(co.envelope, ST_GeomFromEWKT(?), 'T*T***T**') ='TRUE' "
                  + "UNION ALL " +
                  "SELECT co.gmlid, co.class_id " +
                  "FROM CITYOBJECT co " +
                  "WHERE " +
                      "(ST_Relate(co.envelope, ST_GeomFromEWKT(?), 'T*F**F***')
                            ='TRUE' OR " +      // inside and coveredby
                      "ST_Relate(co.envelope, ST_GeomFromEWKT(?), '*TF**F***')
                            ='TRUE' OR " +      // coveredby
                      "ST_Relate(co.envelope, ST_GeomFromEWKT(?), '**FT*F***')
                            ='TRUE' OR " +      // coveredby
                      "ST_Relate(co.envelope, ST_GeomFromEWKT(?), '**F*TF***')
                            ='TRUE') " + // coveredby
                "UNION ALL " +
                  "SELECT co.gmlid, co.class_id " +
                  "FROM CITYOBJECT co " +
                  "WHERE " +
                      "ST_Relate(co.envelope, ST_GeomFromEWKT(?), 'T*F**FFF*')
                            ='TRUE' " +         // equal
                      "ORDER BY 2"; // ORDER BY co.class_id*/

100     //      public static final String QUERY_EXTRUDED_HEIGHTS =
        //              "SELECT " + // "b.measured_height, " +
        //              "SDO_GEOM.SDO_MAX_MBR_ORDINATE(co.envelope, 3) -
                            SDO_GEOM.SDO_MIN_MBR_ORDINATE(co.envelope, 3) AS
                            envelope_measured_height " +
        //              "FROM CITYOBJECT co " + // ", BUILDING b " +
        //              "WHERE " +
        //                      "co.gmlid = ?"; // + " AND b.building_root_id = co.id";
        public static final String GET_EXTRUDED_HEIGHT =
                "SELECT " + // "b.measured_height, " +
                "ST_ZMax(Box3D(co.envelope)) - ST_ZMin(Box3D(co.envelope)) AS
                        envelope_measured_height " +
                "FROM CITYOBJECT co " + // ", BUILDING b " +
                "WHERE co.gmlid = ?"; // + " AND b.building_root_id = co.id";

114     //      public static final String INSERT_GE_ZOFFSET =
        //              "INSERT INTO CITYOBJECT_GENERICATTRIB (ID, ATTRNAME, DATATYPE,
                            STRVAL, CITYOBJECT_ID) " +
        //              "VALUES (CITYOBJECT_GENERICATT_SEQ.NEXTVAL, ?, 1, ?,
        //                      (SELECT ID FROM CITYOBJECT WHERE gmlid = ?))";
        //
        //      public static final String TRANSFORM_GEOMETRY_TO_WGS84 =
        //              "SELECT SDO_CS.TRANSFORM(?, 4326) FROM DUAL";
        //
        //      public static final String TRANSFORM_GEOMETRY_TO_WGS84_3D =
        //              "SELECT SDO_CS.TRANSFORM(?, 4329) FROM DUAL";
        //
        //      public static final String GET_ENVELOPE_IN_WGS84_FROM_GML_ID =
        //              "SELECT SDO_CS.TRANSFORM(co.envelope, 4326) " +
        //              "FROM CITYOBJECT co " +
        //              "WHERE co.gmlid = ?";
        //
        //      public static final String GET_ENVELOPE_IN_WGS84_3D_FROM_GML_ID =
        //              "SELECT SDO_CS.TRANSFORM(co.envelope, 4329) " +
        //              "FROM CITYOBJECT co " +
        //              "WHERE co.gmlid = ?";
```

```
public static final String INSERT_GE_ZOFFSET =
        "INSERT INTO CITYOBJECT_GENERICATTRIB (ID, ATTRNAME, DATATYPE, " +
                "STRVAL, CITYOBJECT_ID) " +
        "VALUES (nextval('CITYOBJECT_GENERICATTRIB_ID_SEQ'), ?, 1, ?, " +
            "(SELECT ID FROM CITYOBJECT WHERE gmlid = ?))";

public static final String TRANSFORM_GEOMETRY_TO_WGS84 =
        "SELECT ST_Transform(?, 4326)";

public static final String TRANSFORM_GEOMETRY_TO_WGS84_3D =
        "SELECT ST_Transform(?, 94329)";

public static final String GET_ENVELOPE_IN_WGS84_FROM_GML_ID =
        "SELECT ST_Transform(co.envelope, 4326) " +
        "FROM CITYOBJECT co " +
        "WHERE co.gmlid = ?";

public static final String GET_ENVELOPE_IN_WGS84_3D_FROM_GML_ID =
        "SELECT ST_Transform(co.envelope, 94329) " +
        "FROM CITYOBJECT co " +
        "WHERE co.gmlid = ?";

591  //    public static final String
wdh  //     BUILDING_GET_AGGREGATE_GEOMETRIES_FOR_LOD2_OR_HIGHER       =
     //           "SELECT sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom,
     //                 <TOLERANCE>)) aggr_geom " +
     //           "FROM (SELECT sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom,
     //                 <TOLERANCE>)) aggr_geom " +
     //           "FROM (SELECT sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom,
     //                 <TOLERANCE>)) aggr_geom " +
     //           "FROM (SELECT sdo_aggr_union(mdsys.sdoaggrtype(simple_geom,
     //                 <TOLERANCE>)) aggr_geom " +
     //           "FROM (" +
     //
     //           "SELECT * FROM (" +
     //           "SELECT * FROM (" +
     //
     //     "SELECT geodb_util.to_2d(sg.geometry, <2D_SRID>) AS simple_geom " +
     //           //"SELECT geodb_util.to_2d(sg.geometry, (select srid from
     //           //     database_srs)) AS simple_geom " +
     //           //"SELECT sg.geometry AS simple_geom " +
     //           "FROM SURFACE_GEOMETRY sg " +
     //           "WHERE " +
     //             "sg.root_id IN( " +
     //                "SELECT b.lod<LoD>_geometry_id " +
     //                "FROM CITYOBJECT co, BUILDING b " +
     //                "WHERE "+
     //                  "co.gmlid = ? " +
     //                  "AND b.building_root_id = co.id " +
     //                  "AND b.lod<LoD>_geometry_id IS NOT NULL " +
     //                "UNION " +
     //                "SELECT ts.lod<LoD>_multi_surface_id " +
     //                "FROM CITYOBJECT co, BUILDING b, THEMATIC_SURFACE ts " +
     //                "WHERE "+
     //                  "co.gmlid = ? " +
     //                  "AND b.building_root_id = co.id " +
     //                  "AND ts.building_id = b.id " +
     //                  "AND ts.lod<LoD>_multi_surface_id IS NOT NULL "+
     //             ") " +
     //             "AND sg.geometry IS NOT NULL" +
     //
     //           ") WHERE sdo_geom.validate_geometry(simple_geom, <TOLERANCE>)
     //                 = 'TRUE'" +
```

```
//            ") WHERE sdo_geom.sdo_area(simple_geom, <TOLERANCE>)
//                  > <TOLERANCE> ) " +
//            "GROUP BY mod(rownum, <GROUP_BY_1>) " +
//            ") " +
//            "GROUP BY mod (rownum, <GROUP_BY_2>) " +
//            ") " +
//            "GROUP BY mod (rownum, <GROUP_BY_3>) " +
//            ")";
"SELECT ST_Union(get_valid_area.simple_geom) " +
"FROM (" +
"SELECT * FROM (" +
   "SELECT * FROM (" +
"SELECT ST_Force_2D(sg.geometry) AS simple_geom " +
      "FROM SURFACE_GEOMETRY sg " +
      "WHERE " +
            "sg.root_id IN( " +
                  "SELECT b.lod<LoD>_geometry_id " +
                  "FROM CITYOBJECT co, BUILDING b " +
                  "WHERE "+
                        "co.gmlid = ? " +
                        "AND b.building_root_id = co.id " +
                        "AND b.lod<LoD>_geometry_id IS NOT NULL " +
                  "UNION " +
                  "SELECT ts.lod<LoD>_multi_surface_id " +
                  "FROM CITYOBJECT co, BUILDING b, THEMATIC_SURFACE ts " +
                  "WHERE "+
                        "co.gmlid = ? " +
                        "AND b.building_root_id = co.id " +
                        "AND ts.building_id = b.id " +
                        "AND ts.lod<LoD>_multi_surface_id IS NOT NULL "+
                  ") " +
            "AND sg.geometry IS NOT NULL) AS get_geoms " +
   "WHERE ST_IsValid(get_geoms.simple_geom) = 'TRUE') AS get_valid_geoms " +
"WHERE ST_Area(get_valid_geoms.simple_geom) > <TOLERANCE>) AS
get_valid_area";  // PostgreSQL-Compiler needs subquery-aliases
```

de.tub.citydb.modules.kml.database.**KmlSplitter**

```
264  //     BoundingBox tile =
wdh  //           exportFilter.getBoundingBoxFilter().getFilterState();
     //     OracleResultSet rs = null;
     //     PreparedStatement spatialQuery = null;
     //     try {
     //           spatialQuery =
     //           connection.prepareStatement(TileQueries.QUERY_GET_GMLIDS);
     //           int srid =
     //           DatabaseConnectionPool.getInstance().
     //           getActiveConnectionMetaData().getReferenceSystem().getSrid();
     //
     //           spatialQuery.setInt(1, srid);
     //           // coordinates for inside
     //           spatialQuery.setDouble(2, tile.getLowerLeftCorner().getX());
     //           spatialQuery.setDouble(3, tile.getLowerLeftCorner().getY());
     //           spatialQuery.setDouble(4, tile.getUpperRightCorner().getX());
     //           spatialQuery.setDouble(5, tile.getUpperRightCorner().getY());
     //           spatialQuery.setInt(6, srid);
     //
     //           // coordinates for overlapbdydisjoint
     //           spatialQuery.setDouble(7, tile.getLowerLeftCorner().getX());
     //           spatialQuery.setDouble(8, tile.getUpperRightCorner().getY());
     //           spatialQuery.setDouble(9, tile.getLowerLeftCorner().getX());
     //           spatialQuery.setDouble(10, tile.getLowerLeftCorner().getY());
     //           spatialQuery.setDouble(11, tile.getUpperRightCorner().getX());
```

```
//     spatialQuery.setDouble(12, tile.getLowerLeftCorner().getY());
//
//     rs = (OracleResultSet)query.executeQuery();
ResultSet rs = null;
PreparedStatement query = null;
String lineGeom = null;
String polyGeom = null;

try {
  if (filterConfig.isSetComplexFilter() &&
       filterConfig.getComplexFilter().getTiledBoundingBox().isSet())
{
       query =connection.prepareStatement(
                     Queries.CITYOBJECTGROUP_MEMBERS_IN_BBOX);
       BoundingBox tile = exportFilter.getBoundingBoxFilter()
                                                .getFilterState();
       int srid = dbSrs.getSrid();

       lineGeom = "SRID=" + srid + ";LINESTRING(" +
              tile.getLowerLeftCorner().getX() + " " +
              tile.getUpperRightCorner().getY() + "," +
              tile.getLowerLeftCorner().getX() + " " +
              tile.getLowerLeftCorner().getY() + "," +
              tile.getUpperRightCorner().getX() + " " +
              tile.getLowerLeftCorner().getY() + ")'";

       polyGeom = "SRID=" + srid + ";POLYGON((" +
              tile.getLowerLeftCorner().getX() + " " +
              tile.getLowerLeftCorner().getY() + "," +
              tile.getLowerLeftCorner().getX() + " " +
              tile.getUpperRightCorner().getY() + "," +
              tile.getUpperRightCorner().getX() + " " +
              tile.getUpperRightCorner().getY() + "," +
              tile.getUpperRightCorner().getX() + " " +
              tile.getLowerLeftCorner().getY() + "," +
              tile.getLowerLeftCorner().getX() + " " +
              tile.getLowerLeftCorner().getY() + "))";

       query.setString(1, lineGeom);
       query.setString(2, polyGeom);
       query.setString(3, polyGeom);
       query.setString(4, polyGeom);
       query.setString(5, polyGeom);
       query.setString(6, polyGeom);

       rs = query.executeQuery();
```

Das folgende Beispiel aus der Klasse `BallonTemplateHandlerImpl` enthält Abfragen mit so genannten „Window“ Funktionen. Mit Window Funktionen werden Berechnungen auf die Spalten eines `ResultSet` ausgeführt ähnlich wie bei Aggregationsfunktionen wie Count, SUM oder AVG. Allerdings wird das Ergebnis bei Window Funktionen nicht auf eine Zeile reduziert. Die hier angewandte Funktion `row_number()` nummeriert die Zeilen des `ResultSet` mit dem Ziel den ersten Wert auszuwählen, da die Sprechblasen einer KML-Datei („Balloons“) nur einen Wert pro Attribut enthalten sollen.

de.tub.citydb.modules.kml.database.**BalloonTemplateHandlerImpl**

```
1152   sqlStatement = sqlStatement + ") AS subquery";
wdh    // PostgreSQL-Query needs an alias here
```

```
1206  //     sqlStatement = "SELECT * FROM " +
      //             " (SELECT a.*, ROWNUM rnum FROM (" + sqlStatement +
      //             " ORDER by " + tableShortId + "." + columns.get(0) + " ASC) a"
      //             + " WHERE ROWNUM <= " + rownum + ") "
      //             + "WHERE rnum >= " + rownum;
      sqlStatement = "SELECT * FROM " +
      "(SELECT sqlstat.*, ROW_NUMBER() OVER(ORDER BY sqlstat.* ASC) AS rnum" +
             " FROM (" + sqlStatement +
             " ORDER BY " + tableShortId + "." + columns.get(0) + " ASC) sqlstat)
             AS subq WHERE rnum = " + rownum;

      //     else if (FIRST.equalsIgnoreCase(aggregateFunction)) {
      //             sqlStatement = "SELECT * FROM (" + sqlStatement +
      //             " ORDER by " + tableShortId + "." + columns.get(0) + " ASC)" +
      //             " WHERE ROWNUM = 1";
      //     }
      //     else if (LAST.equalsIgnoreCase(aggregateFunction)) {
      //             sqlStatement = "SELECT * FROM (" + sqlStatement +
      //             " ORDER by " + tableShortId + "." + columns.get(0) + " DESC)"
      //             + " WHERE ROWNUM = 1";
      //     }
      else if (FIRST.equalsIgnoreCase(aggregateFunction)) {
             sqlStatement = "SELECT * FROM " +
             "(SELECT sqlstat.*, ROW_NUMBER() OVER(ORDER BY sqlstat.* ASC)
             AS rnum FROM (" + sqlStatement +
             " ORDER BY " + tableShortId + "." + columns.get(0) + " ASC) sqlstat)
             AS subq WHERE rnum = 1";
      }
      else if (LAST.equalsIgnoreCase(aggregateFunction)) {
             sqlStatement = "SELECT * FROM " +
             "(SELECT sqlstat.*, ROW_NUMBER() OVER(ORDER BY sqlstat.* ASC)
             AS rnum FROM (" + sqlStatement +
             " ORDER BY " + tableShortId + "." + columns.get(0) + " DESC)
             sqlstat) AS subq WHERE rnum = 1";
      }
```

B.9.2. Geometrien für KML Plcemarks

de.tub.citydb.modules.kml.database.**CityObjectGroup**

```
189   //     STRUCT buildingGeometryObj = (STRUCT)rs.getObject(1);
      PGgeometry pgBuildingGeometry = (PGgeometry)rs.getObject(1);

201   //     JGeometry groundSurface =
wdh+  //     convertToWGS84(JGeometry.load(buildingGeometryObj));
      //     int dim = groundSurface.getDimensions();
      //     for (int i = 0; i < groundSurface.getElemInfo().length; i = i+3) {
      //             LinearRingType linearRing = kmlFactory.createLinearRingType();
      //             BoundaryType boundary = kmlFactory.createBoundaryType();
      //             boundary.setLinearRing(linearRing);
      //             switch (groundSurface.getElemInfo()[i+1]) {
      //                     case EXTERIOR_POLYGON_RING:      // = 1003
      //                             polygon.setOuterBoundaryIs(boundary);
      //                                     break;
      //                     case INTERIOR_POLYGON_RING:      // = 2003
      //                             polygon.getInnerBoundaryIs().add(boundary);
      //                             break;
      //                     case POINT:                      // = 1
      //                     case LINE_STRING:                // = 2
      //                             continue;
```

```
//      spatialQuery.setDouble(12, tile.getLowerLeftCorner().getY());
//                  default:
//                          Logger.getInstance().warn("Unknown
                                    geometry for " + work.getGmlId());
//                          continue;
//              }
//      double[] ordinatesArray = groundSurface.getOrdinatesArray();
//      int startNextGeometry =((i+3)< groundSurface.getElemInfo().length) ?
//              groundSurface.getElemInfo()[i+3]- 1: // still more geometries
//                  ordinatesArray.length;              // default
//
//      // order points counter-clockwise
//      for (int j = startNextGeometry - dim;
                j >= groundSurface.getElemInfo()[i] - 1; j = j dim) {
//              linearRing.getCoordinates().add(String.valueOf(
                ordinatesArray[j] + "," + ordinatesArray[j+1] + ",0"));
//      }
Geometry groundSurface = convertToWGS84(pgBuildingGeometry.getGeometry());

switch (groundSurface.getSubGeometry(i).getType()) {
        case POLYGON:
          Polygon polyGeom = (Polygon)groundSurface;

          for (int ring = 0; ring < polyGeom.numRings(); ring++){
            LinearRingType linearRing = kmlFactory.createLinearRingType();
            BoundaryType boundary = kmlFactory.createBoundaryType();
            boundary.setLinearRing(linearRing);

            double [] ordinatesArray =
              new double[polyGeom.getRing(ring).numPoints()*2];

            for (int j=polyGeom.getRing(ring).numPoints()-1,k=0;
              j>=0;j--,k+=2){
              ordinatesArray[k] = polyGeom.getRing(ring).getPoint(j).x;
              ordinatesArray[k+1] = polyGeom.getRing(ring).getPoint(j).y;
            }

            // the first ring usually is the outer ring in a PostGIS-Polygon
            // e.g. POLYGON((outerBoundary),(innerBoundary),(innerBoundary))
            if (ring == 0){
              polygon.setOuterBoundaryIs(boundary);
              for (int j = 0; j < ordinatesArray.length; j+=2) {
                linearRing.getCoordinates().add
                  (String.valueOf(ordinatesArray[j] + "," +
                            ordinatesArray[j+1] + ",0"));
              }
            }
            else {
              polygon.getInnerBoundaryIs().add(boundary);
              for (int j = ordinatesArray.length - 2; j >= 0; j-=2) {
                linearRing.getCoordinates().add(
                  String.valueOf(ordinatesArray[j] + "," +
                    ordinatesArray[j+1] + ",0"));
              }
            }
          }
          break;
        case Geometry.MULTIPOLYGON:
          MultiPolygon multiPolyGeom = (MultiPolygon) groundSurface;
          multiPolygon = new PolygonType[multiPolyGeom.numPolygons()];

          for (int p = 0; p < multiPolyGeom.numPolygons(); p++){
              Polygon subPolyGeom = multiPolyGeom.getPolygon(p);
```

```
                    multiPolygon[p] = kmlFactory.createPolygonType();
                    multiPolygon[p].setTessellate(true);
                    multiPolygon[p].setExtrude(true);
                    multiPolygon[p].setAltitudeModeGroup
                    (kmlFactory.createAltitudeMode(
                                AltitudeModeEnumType.RELATIVE_TO_GROUND));

                for (int ring = 0; ring < subPolyGeom.numRings(); ring++){
                    LinearRingType linearRing =
                                    kmlFactory.createLinearRingType();
                    BoundaryType boundary = kmlFactory.createBoundaryType();
                    boundary.setLinearRing(linearRing);

                    double [] ordinatesArray = new
                    double[subPolyGeom.getRing(ring).numPoints() * 2];

                    for (int j=subPolyGeom.getRing(ring).numPoints()-1, k=0;
                        j >= 0; j--, k+=2){
                        ordinatesArray[k] =
                            subPolyGeom.getRing(ring).getPoint(j).x;
                        ordinatesArray[k+1] =
                            subPolyGeom.getRing(ring).getPoint(j).y;
                    }
                    // the first ring usually is the outer ring in a PostGIS-
                    // Polygon e.g. POLYGON((outerBoundary),(innerBoundary),
                    // (innerBoundary))
                    if (ring == 0){
                        multiPolygon[p].setOuterBoundaryIs(boundary);
                        for (int j = 0; j < ordinatesArray.length; j+=2) {
                            linearRing.getCoordinates().add(
                            String.valueOf(ordinatesArray[j] + "," +
                            ordinatesArray[j+1] + ",0"));
                        }
                    }
                    else {
                        multiPolygon[p].getInnerBoundaryIs().add(boundary);
                        for (int j = ordinatesArray.length-2;j>=0;j-=2) {
                            linearRing.getCoordinates().add(
                            String.valueOf(ordinatesArray[j] + "," +
                            ordinatesArray[j+1] + ",0"));
                        }
                    }
                }
            }
            case Geometry.POINT:
            case Geometry.LINESTRING:
            case Geometry.MULTIPOINT:
            case Geometry.MULTILINESTRING:
            case Geometry.GEOMETRYCOLLECTION:
                continue;
            default:
                Logger.getInstance().warn("Unknown geometry for " +
                    work.getGmlId());
                continue;
            }
        }
        if (polygon != null){
            multiGeometry.getAbstractGeometryGroup().add(
                kmlFactory.createPolygon(polygon));
        }
        if (multiPolygon != null){
            for (int p = 0; p < multiPolygon.length; p++){
                multiGeometry.getAbstractGeometryGroup().add(
                kmlFactory.createPolygon(multiPolygon[p]));
```

de.tub.citydb.modules.kml.database.**KmlGenericObject**

```
2031  //     STRUCT buildingGeometryObj = (STRUCT)rs.getObject(1);
wdh   //     JGeometry surface =
      //             convertToWGS84(JGeometry.load(buildingGeometryObj));
      //     double[] ordinatesArray = surface.getOrdinatesArray();
      PGgeometry pgBuildingGeometry = (PGgeometry)rs.getObject(1);
      Polygon surface =
             (Polygon)convertToWGS84(pgBuildingGeometry.getGeometry());

      double[] ordinatesArray = new double[surface.numPoints()*3];
      for (int i = 0, j = 0; i < surface.numPoints(); i++, j+=3){
             ordinatesArray[j] = surface.getPoint(i).x;
             ordinatesArray[j+1] = surface.getPoint(i).y;
             ordinatesArray[j+2] = surface.getPoint(i).z;
      }

2064  //     for (int i = 0; i < surface.getElemInfo().length; i = i+3) {
wdh   //             LinearRingType linearRing = kmlFactory.createLinearRingType();
      //             BoundaryType boundary = kmlFactory.createBoundaryType();
      //             boundary.setLinearRing(linearRing);
      //             if (surface.getElemInfo()[i+1] == EXTERIOR_POLYGON_RING) {
      //                     polygon.setOuterBoundaryIs(boundary);
      //             }
      //             else { // INTERIOR_POLYGON_RING
      //                     polygon.getInnerBoundaryIs().add(boundary);
      //             }
      //             int startNextRing = ((i+3) <     surface.getElemInfo().length) ?
      //                     surface.getElemInfo()[i+3] - 1: // still holes to come
      //                             ordinatesArray.length; // default
      //             // order points clockwise
      //             for (int j = surface.getElemInfo()[i] - 1; j < startNextRing;
      //                     j = j+3) {
      //                     linearRing.getCoordinates().add(
      //                        String.valueOf(
      //                            reducePrecisionForXorY(ordinatesArray[j]) + "," +
      //                            reducePrecisionForXorY(ordinatesArray[j+1]) +","+
      //                            reducePrecisionForZ(ordinatesArray[j+2] +
      //                            zOffset)));
      //                     probablyRoof = ...
      int cellCount = 0; // equivalent to first value of Oracle's SDO_ELEM_INFO
      for (int i = 0; i < surface.numRings(); i++){
             LinearRingType linearRing = kmlFactory.createLinearRingType();
             BoundaryType boundary = kmlFactory.createBoundaryType();
             boundary.setLinearRing(linearRing);
             if (i == 0) {       // first ring is the outer ring
                     polygon.setOuterBoundaryIs(boundary);
             } else {
                     polygon.getInnerBoundaryIs().add(boundary);
             }
             int startNextRing = ((i+1) < surface.numRings()) ?
                     (surface.getRing(i).numPoints()*3): // still holes to come
                             ordinatesArray.length; // default
             // order points clockwise
             for (int j = cellCount; j < startNextRing; j+=3 {
               linearRing.getCoordinates().add(
                 String.valueOf(
                    reducePrecisionForXorY(ordinatesArray[j]) + "," +
                    reducePrecisionForXorY(ordinatesArray[j+1]) + "," +
                    reducePrecisionForZ(ordinatesArray[j+2] + zOffset)))
              probablyRoof = ...
        }
        cellCount += (surface.getRing(i).numPoints()*3);
        }
```

```
2540  //     int contourCount = unconvertedSurface.getElemInfo().length/3;
      //     // remove normal-irrelevant points
      //     int startContour1 = unconvertedSurface.getElemInfo()[0] - 1;
      //     int endContour1 = (contourCount == 1) ?
      //      ordinatesArray.length: // last
      //         unconvertedSurface.getElemInfo()[3] - 1; // holes are irrelevant
      //                                                   for normal calculation
      //     // last point of polygons in gml is identical to first and useless for
      //     // GeometryInfo
      //     endContour1 = endContour1 - 3;
      int contourCount = unconvertedSurface.numRings();
      int startContour1 = 0;
      int endContour1 = (contourCount == 1) ?
             ordinatesArray.length: // last
                    (unconvertedSurface.getRing(startContour1).numPoints()*3);
                          endContour1 = endContour1 - 3;

2586  //     for (int i = 0; i < ordinatesArray.length; i = i + 3) {
      //       // coordinates = coordinates + hlDistance * (dot product of normal
              // vector and unity vector)
      //       ordinatesArray[i] = ordinatesArray[i] + hlDistance * nx;
      //       ordinatesArray[i+1] = ordinatesArray[i+1] + hlDistance * ny;
      //       ordinatesArray[i+2] = ordinatesArray[i+2]+zOffset+hlDistance*nz;
      //     }

      for (int i = 0, j = 0; i < unconvertedSurface.numPoints(); i++, j+=3){
        unconvertedSurface.getPoint(i).x = ordinatesArray[j] + hlDistance*nx;
        unconvertedSurface.getPoint(i).y = ordinatesArray[j+1] + hlDistance*ny;
        unconvertedSurface.getPoint(i).z = ordinatesArray[j+2] + zOffset +
                                                    hlDistance * nz;
      }
```

B.9.3. Texturen für den COLLADA-Export

In der *3DCityDB* können Texturen in Bildformaten vorliegen, die für Methoden von ORDImage unbekannt sind. Im *KML-Exporter* wurden daher zwei Wege implementiert, um alle Formate abdecken zu können: Einer für ORDImages und ein Alternativer für BLOBs. Glücklicherweise konnte letzterer für die *PostGIS*-Version verwendet werden. In folgenden vier Klassen war es ausreichend die ORDImage Methoden auszuklammern:

de.tub.citydb.modules.kml.concurrent.**KmlExportWorker**
de.tub.citydb.modules.kml.controller.**KmlExporter**
de.tub.citydb.modules.kml.database.**ColladaBundle**
de.tub.citydb.modules.kml.database.**KmlExporterManager**

de.tub.citydb.modules.kml.database.**KmlGenericObject**

```
2238  //     OrdImage texImage = null;
      InputStream texImage = null;

2262  addTexImageUri(surfaceId, texImageUri);
      // if (getTexOrdImage(texImageUri) == null) { // not already marked as
                                                    wrapping texture
```

```
2283  //bufferedImage = ImageIO.read(texImage.getDataInStream());
       bufferedImage = ImageIO.read(texImage);

2290   // else {
       //         addTexOrdImage(texImageUri, texImage);
       //       }
       // }

2256   // texture wrapping -- it conflicts with texture atlas
       removeTexImage(texImageUri);
       BufferedImage bufferedImage = null;
          try {
              bufferedImage = ImageIO.read(texImage);
          } catch (IOException e) {}
          addTexImage(texImageUri, bufferedImage);
          // addTexOrdImage(texImageUri, texImage);
```

Anhang C

Dokumentation der Testergebnisse

Auf den folgenden Seiten werden die dokumentieren Zeiten für Im- und Export mit beiden Versionen sowie der Speicherplatzbedarf der Datensätze im jeweiligen DBMS in Tabellen aufgelistet. Die Testergebnisse wurden bereits in Kapitel 8 in Diagrammen visualisiert und in im Verhältnis zu mehreren Einflussfaktoren gesetzt (siehe Tabellen 7 bis 11). Dazu gehörten die zeitlichen Veränderungen bei variierender Threadanzahl, Datensatzgröße und flächenhafter Ausdehnung der Datensätze sowie die zeitliche Differenz bei Hinzunahme von Texturen und aktiven Indizes während des Imports.

Tab. C1: Zeiten beim Import von Datensätzen

I M P O R T **von Datensätzen**				**Oracle**					**PostgreSQL / PostGIS**				
CityGML-Datei	**MB**	**Tex**	**Idx**	**1**	**4**	**8**	**16**	**32**	**1**	**4**	**8**	**16**	**32**
Berlin-Tiergarten Süd: 1,15 km², 704 Gebäude 18885 Geometrien 2085 Appearances	16		▲	0:28	0:25	0:21	0:25	0:31	0:17	0:08	0:06	0:07	0:08
				0:14	0:11	0:07	0:08	0:13	0:15	0:06	0:06	0:07	0:08
	78	🖼	▲	5:41	1:36	1:10	1:15	1:14	5:24	4:34	0:49	3:30	3:48
		🖼		5:55	1:20	1:07	1:00	1:02	4.50	3:02	0:51	3:00	3:12
Berlin-Neukölln: 21 km², 11013 Gebäude, 203198 Geometrien, 32062 Appearances	190		▲	6:51	5:26	5:38	7:37	9:14	3:28	0:55	3:24	0:54	1:05
				3:44	1:38	1:37	1:19	1:40	2:02	0:58	2:58	1:01	0:57
	724	🖼	▲	52:36	18:29	16:22	error	error	27:59	16:30	13:39	14:29	22:16
		🖼		43:56	13:07	9:43	error	error	21:58	9:58	9:43	10:20	15:23
Berlin-Spandau: 96,64 km², 48377 Gebäude, 860019 Geometrien, 131662 Appearances	849		▲		40:33	42:45	47:25	47:23		33:33	16:09	17:07	19:37
					7:22	6:25	6:59	7:54		12:21	11:40	11:16	11:48
	1880	🖼	▲		1:50:46	1:48:04	1:46:47	1:52:54		1:45:00	1:11:14	1:08:56	1:38:22
		🖼			54:36	42:25	43:27	44:59		38:22	35:50	39:40	39:44
Berlin-Ring: 136 km², 72585 Gebäude, 2247381 Geometrien, 211737 Appearances	1810		▲		2:02:43	2:12:15	2:24:40	2:33:58		31:38	31:33	37:28	50:27
					13:34	13:25	15:08	18:09		16:22	15:43	16:19	18:24
	6520	🖼	▲		5:44:24	5:35:59	5:00:11	error		3:27:15	3:30:59	3:46:20	3:55:59
		🖼			3:06:03	2:16:17	2:19:31	error		1:27:15	2:09:36	2:16:14	2:40:25
Berlin: 1596 km², 534357 Gebäude, 9083266 Geometrien, 458721 Appearances	8690		▲		23:16:11	23:38:57	20:06:08			3:27:15	3:24:45	3:31:29	
					1:21:42	1:21:42	1:32:51			1:27:15	1:22:47	1:23:48	
	22600	🖼	▲			36:04:16					18:10:36		
		🖼				10:20:26					10:06:01		

🖼 = mit Texturen, ▲ = mit aktivierten Indizes, Zeit = hh.mm.ss

Tab. C2: Zeiten beim Export von Datensätzen

E X P O R T **von Datensätzen**			**Oracle**					**PostgreSQL / PostGIS**				
CityGML-Datei	**MB** (Ora/PgSQL)	**Tex**	**1**	**4**	**8**	**16**	**32**	**1**	**4**	**8**	**16**	**32**
Berlin-Tiergarten Süd: 1,15 km²	18/18		0:27	0:06	0:04	0:04	0:06	1:55	1:43	1:42	1:41	1:43
	82/82		1:25	0:50	0:35	0:30	0:34	2:22	2:08	1:56	1:55	1:52
Berlin-Neukölln: 21 km²	188/194		5:02	0:50	0:35	0:30	0:34	3:56	2:19	2:08	2:04	2:03
	719/724		18:36	11:39	13:37	10:28	11:28	14:11	13:33	13:16	13:11	12:58
Berlin-Spandau: 96,64 km²	824/849			9:00	3:26	3:06	3:15		29:40	28:45	28:22	27:59
	1850/1880			35:46	27:16	26:50	26:16		42:37	37:15	34:38	31:58
Berlin-Ring: 136 km²	1770/1810			17:36	7:30	5:11	5:33		1:02:59	1:01:42	1:01:15	1:01:37
	6570/6520			1:10:23	1:33:17	1:38:09	18:09		1:38:12	1:19:29	1:19:23	1:18:48
Berlin: 1596 km²	8450/8690			1:32:22	1:33:17	1:38:09	1:29:39		3:24:26	3:08:44	3:04:33	3:05:42
	22300/22600			9:55:14	7:40:41	6:57:23	8:33:25		5:34:35	5:22:12	5:20:23	5:30:38

= mit Texturen, Zeit = hh.mm.ss

Tab. C3: Speicherplatzbedarf in der Datenbank in MB

Speicherbedarf **im DBMS in MB**				**Oracle**					**PostgreSQL / PostGIS**				
CityGML-Datei	**MB**	**Tex**	**Idx**	**1**	**4**	**8**	**16**	**32**	**1**	**4**	**8**	**16**	**32**
Berlin-Tiergarten Süd: 1,15 km², 704 Gebäude 18885 Geometrien 2085 Appearances	16		▲	53	54	54	54	54	30	31	31	31	31
				53	53	53	53	54	30	30	30	30	31
	78	[Tex]	▲	143	144	144	145	144	100	101	103	102	102
		[Tex]		142	142	143	142	143	99	99	99	99	99
Berlin-Neukölln: 21 km², 11013 Gebäude, 203198 Geometrien, 32062 Appearances	190		▲	229	241	241	243	248	208	213	213	215	216
				227	239	250	242	247	204	222	222	210	212
	724	[Tex]	▲	1049	1067	1063	error	error	834	834	835	838	846
		[Tex]		1046	1049	1058	error	error	819	817	819	820	836
Berlin-Spandau: 96,64 km², 48377 Gebäude, 860019 Geometrien, 131662 Appearances	849		▲		930	956	957	941		958	934	1002	974
					905	990	924	1043		920	954	978	947
	1880	[Tex]	▲		2659	2669	2667	2683		2213	2289	2293	2281
		[Tex]			2602	2606	2678	2741		2245	2263	2247	2248
Berlin-Ring: 136 km², 72585 Gebäude, 2247381 Geometrien, 211737 Appearances	1810		▲		1868	1926	1961	1999		1841	1761	1937	1782
					1864	1863	1866	2006		1772	1743	1767	1813
	6520	[Tex]	▲		8751	9243	9225	error		7383	7395	7399	7401
		[Tex]			9179	8987	8916	error		7233	7240	7254	7319
Berlin: 1596 km², 534357 Gebäude, 9083266 Geometrien, 458721 Appearances	8690		▲		8219	8291	8345			8281	8321	8335	
					8207	8208	8374			8163	8226	8174	
	22600	[Tex]	▲			29916					23400		
		[Tex]				29618					22100		

[Tex] = mit Texturen, ▲ = mit aktivierten Indizes, Zeit = hh.mm.ss

Anhang D

Weitere Abbildungen

Auf den folgenden Seiten werden Screenshots von den verschiedenen Modulen des *Importer/Exporter* einen Überblick von den Funktionalitäten der Software vermitteln. Weitere Screenshots von Stadtmodellen visualisiert in CityGML-Vierwern und *Google Earth* sollen dem Leser zeigen, wie die CityGML-Daten aussehen können.

D.1. Screenshots vom Importer/Exporter

Abb. D1: Splash-Screen der PostGIS-Version des Importer/Exporter

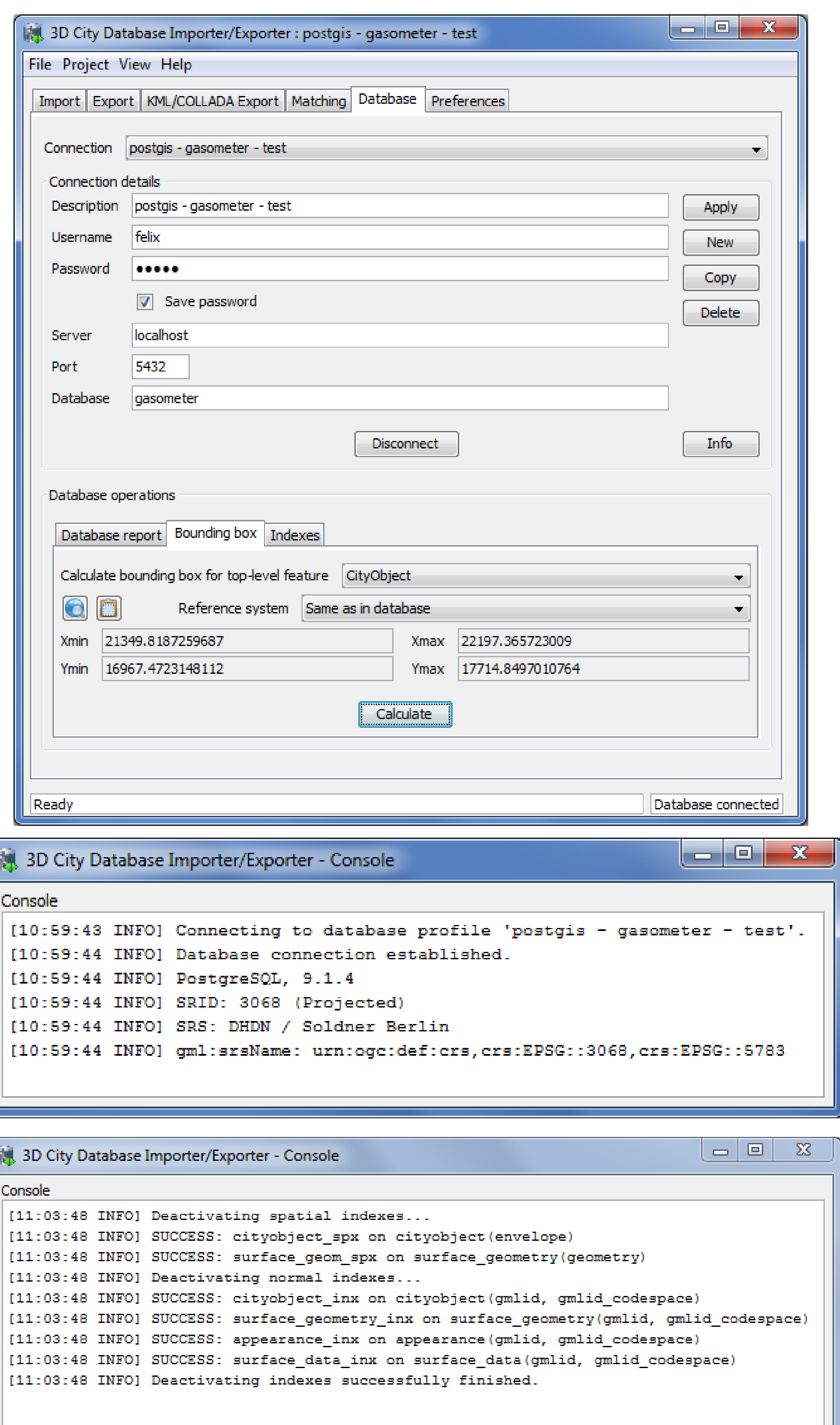


Abb. D2: Datenbank-Panel des Importer/Exporter mit Log-Beispielen für das erfolgreiche Verbinden mit der Datenbank und Rückmeldungen beim Deaktivieren der Indizes

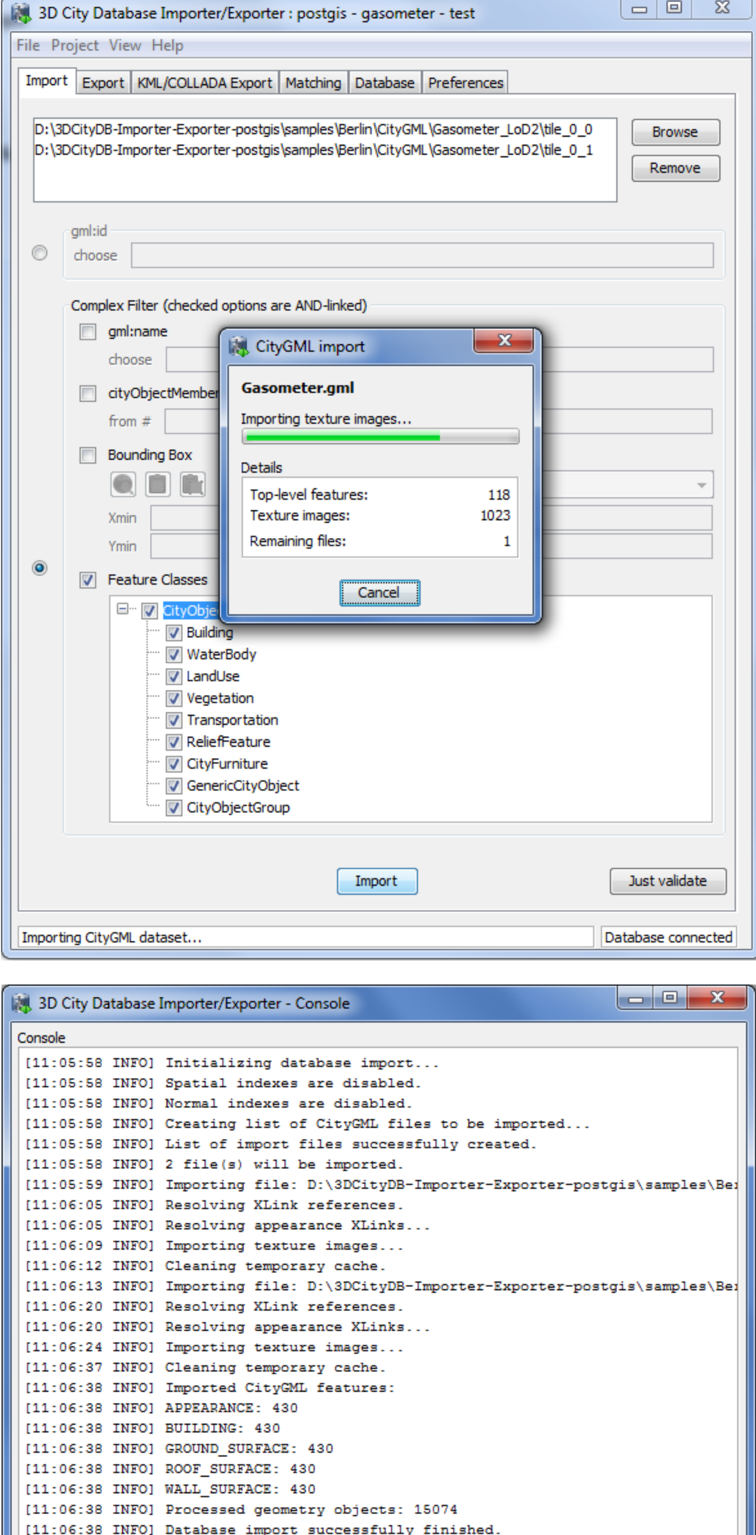
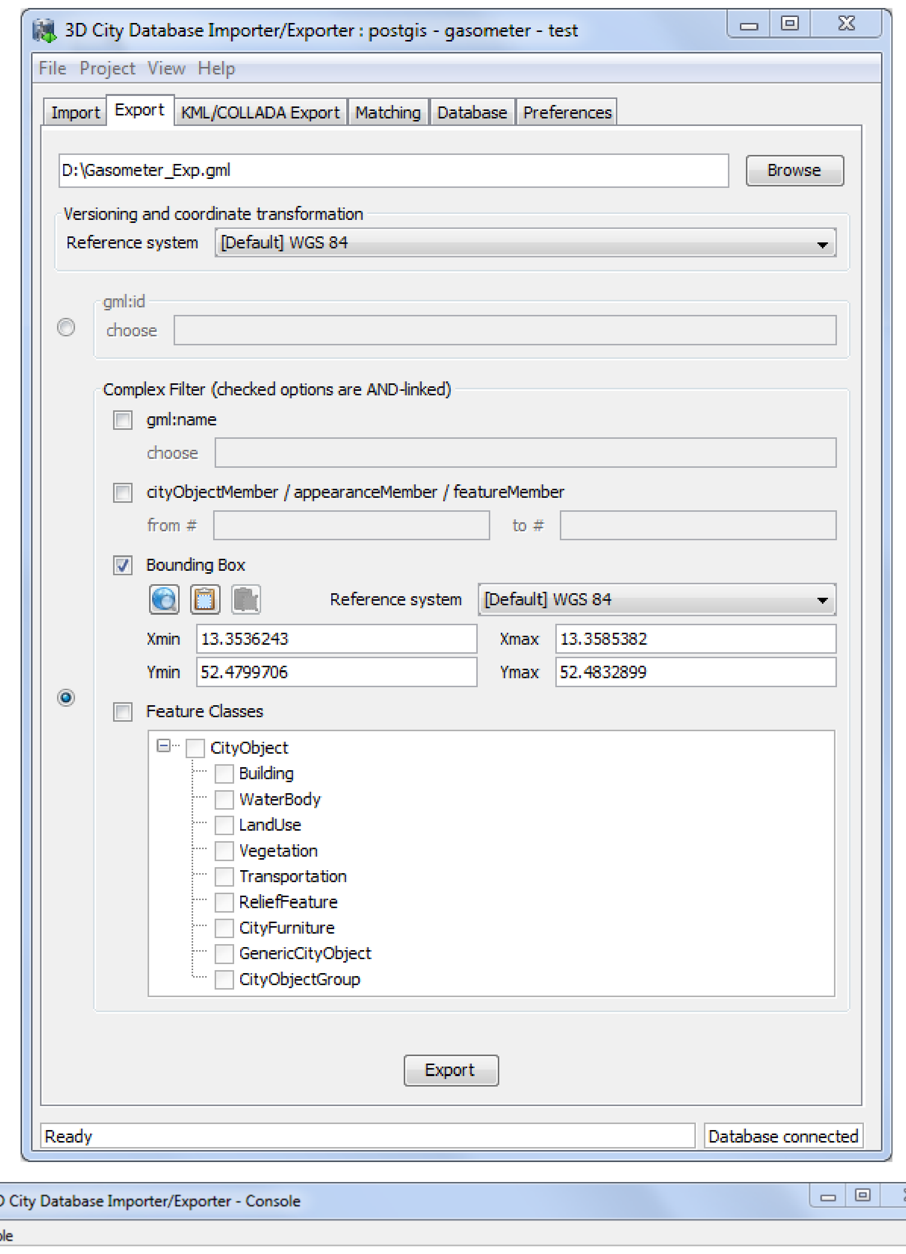


Abb. D3: Import-Panel während eines Importes mit Log-Eintrag im Konsolenfenster für einen erfolgreich abgeschlossenen Import



3D City Database Importer/Exporter - Console

Console

```
[11:20:29 INFO] Initializing database export...
[11:20:29 INFO] Transforming geometry representation to reference system '[Default] WGS 84' (SRID: 4326).
[11:20:29 WARN] Transformation is NOT applied to height reference system.
[11:20:29 INFO] Spatial indexes are enabled.
[11:20:30 INFO] Normal indexes are enabled.
[11:20:30 INFO] Created texture files subfolder 'appearance'.
[11:20:30 INFO] Exporting to file: D:\Gasometer_Exp.gml
[11:20:34 INFO] Processing CityObjectGroup features.
[11:20:35 INFO] Cleaning temporary cache.
[11:20:35 INFO] Exported CityGML features:
[11:20:35 INFO] APPEARANCE: 77
[11:20:35 INFO] BUILDING: 77
[11:20:35 INFO] GROUND_SURFACE: 77
[11:20:35 INFO] ROOF_SURFACE: 77
[11:20:35 INFO] WALL_SURFACE: 77
[11:20:35 INFO] Processed geometry objects: 2907
[11:20:35 INFO] Database export successfully finished.
```

Abb. D4: Export-Panel mit Log-Eintrag im Konsolenfenster für einen erfolgreich abgeschlossenen Export

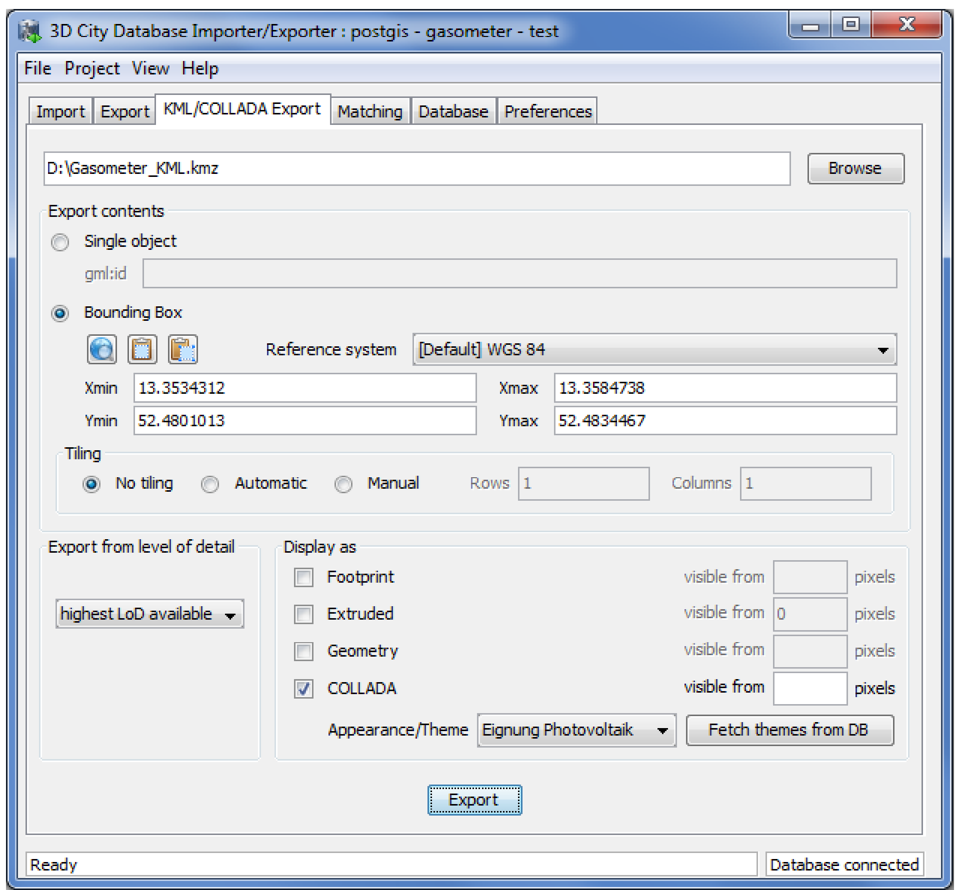


Abb. D5: Das KML-Exporter-Panel

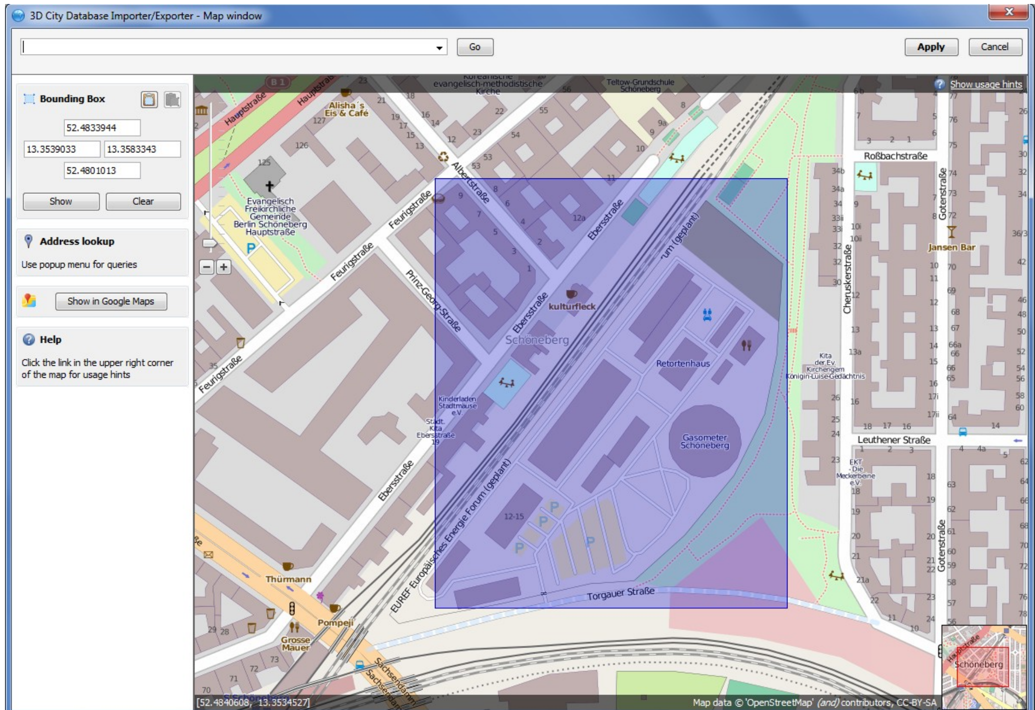


Abb. D6: Das MapWindow zum Betrachten und Editieren des gegenwärtigen Bounding Box Filters (angewendet für Export siehe S.293)

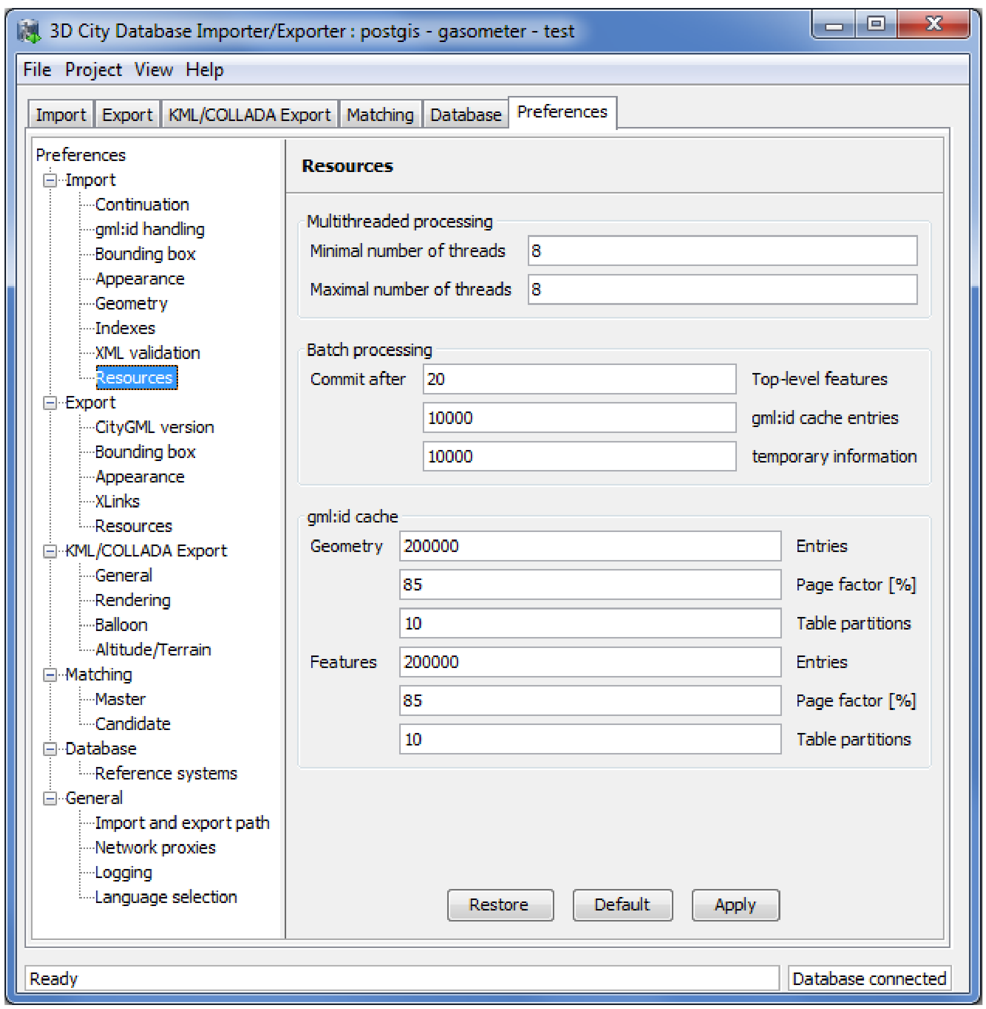


Abb. D7: Einstellungsmöglichkeiten für die verschiedenen Module des Importer/Exporter (abgebildet ist u.a. die Angabe der Anzahl an Ausführungsthreads)

D.2. Screenshots von CityGML-Viewern

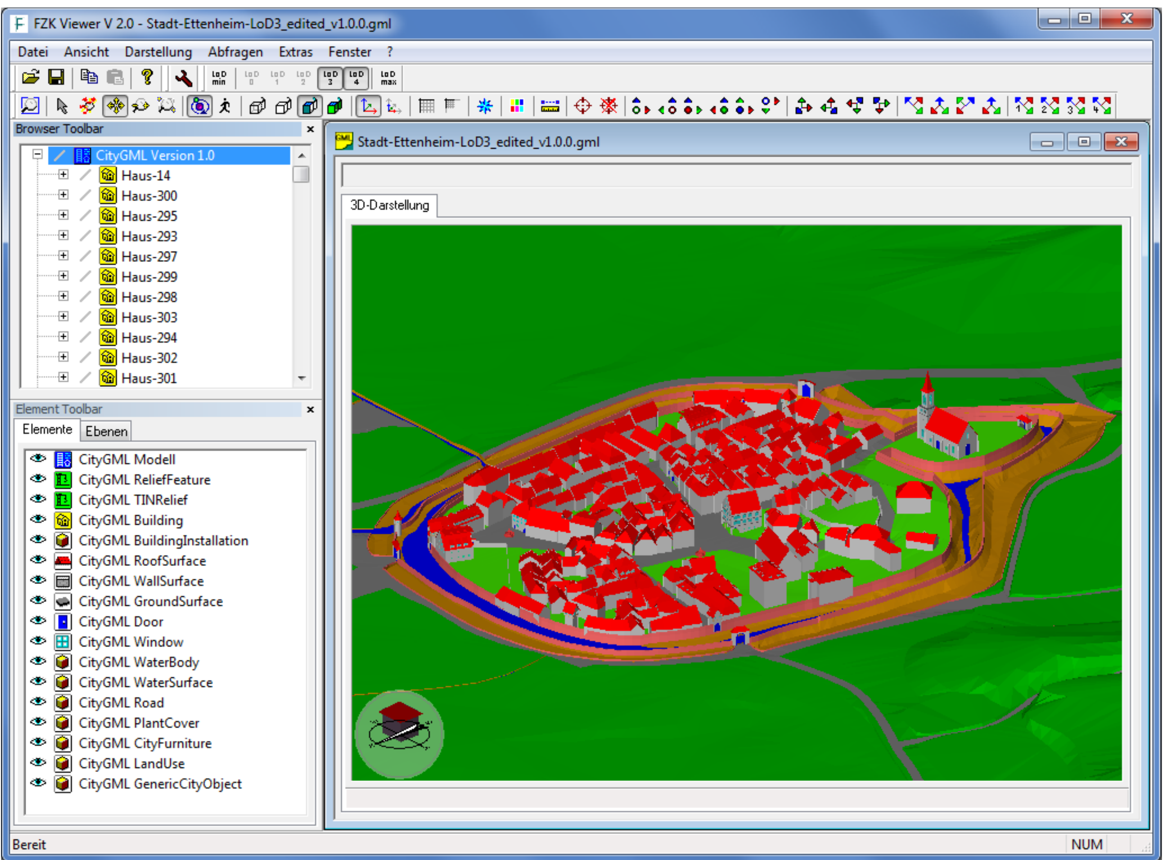


Abb. D8: Datensatz Stadt Ettenheim (verfügbar unter citygml.org) im FZKViewer

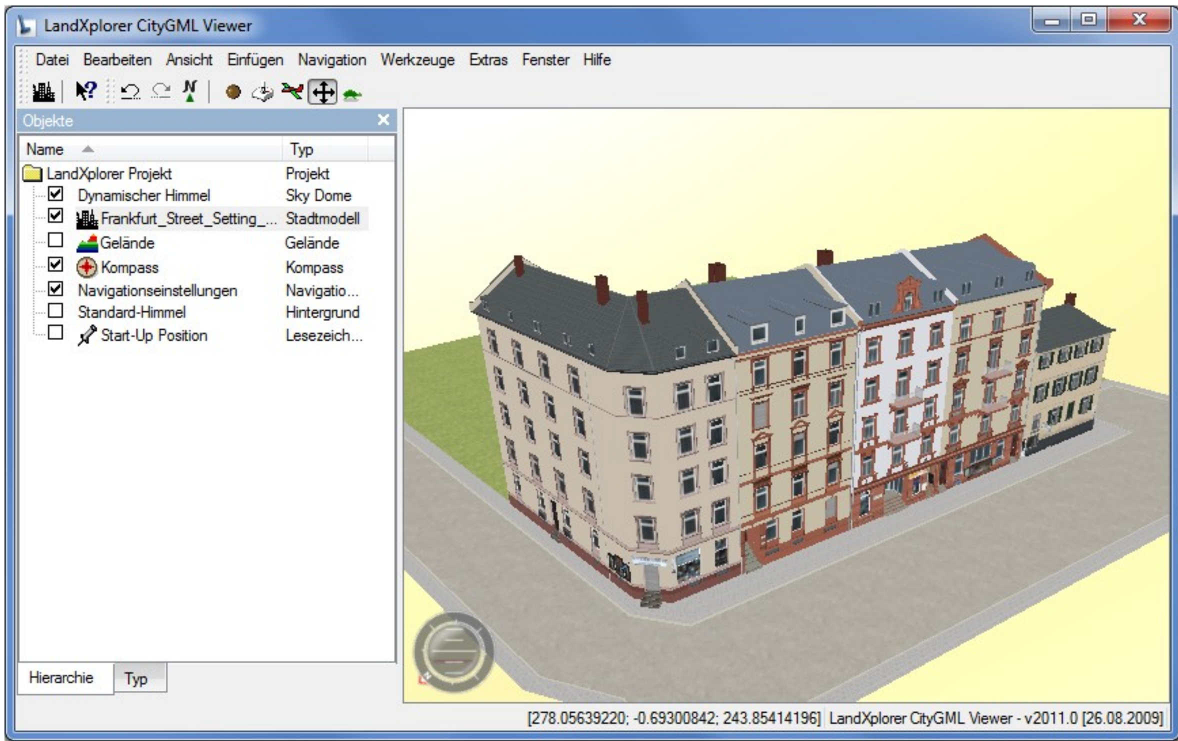


Abb. D9: Datensatz Frankfurt (verfügbar unter citygml.org) im LandXplorer

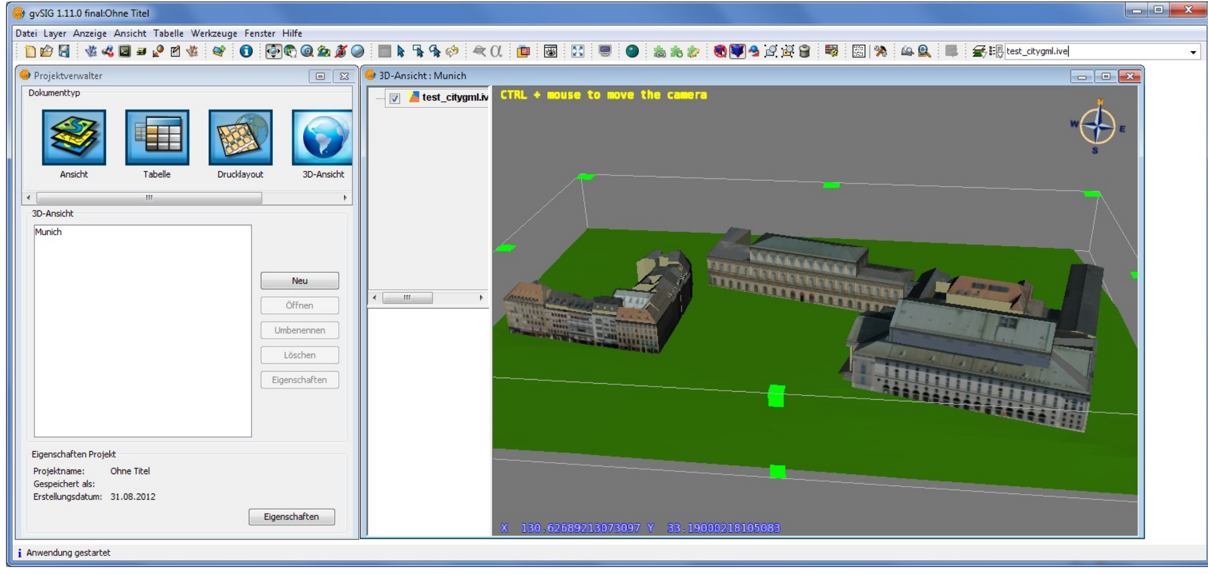


Abb. D10: Datensatz München (verfügbar unter citygml.org) in gvSIG mit 3D-Erweiterung und CityGML-Patch

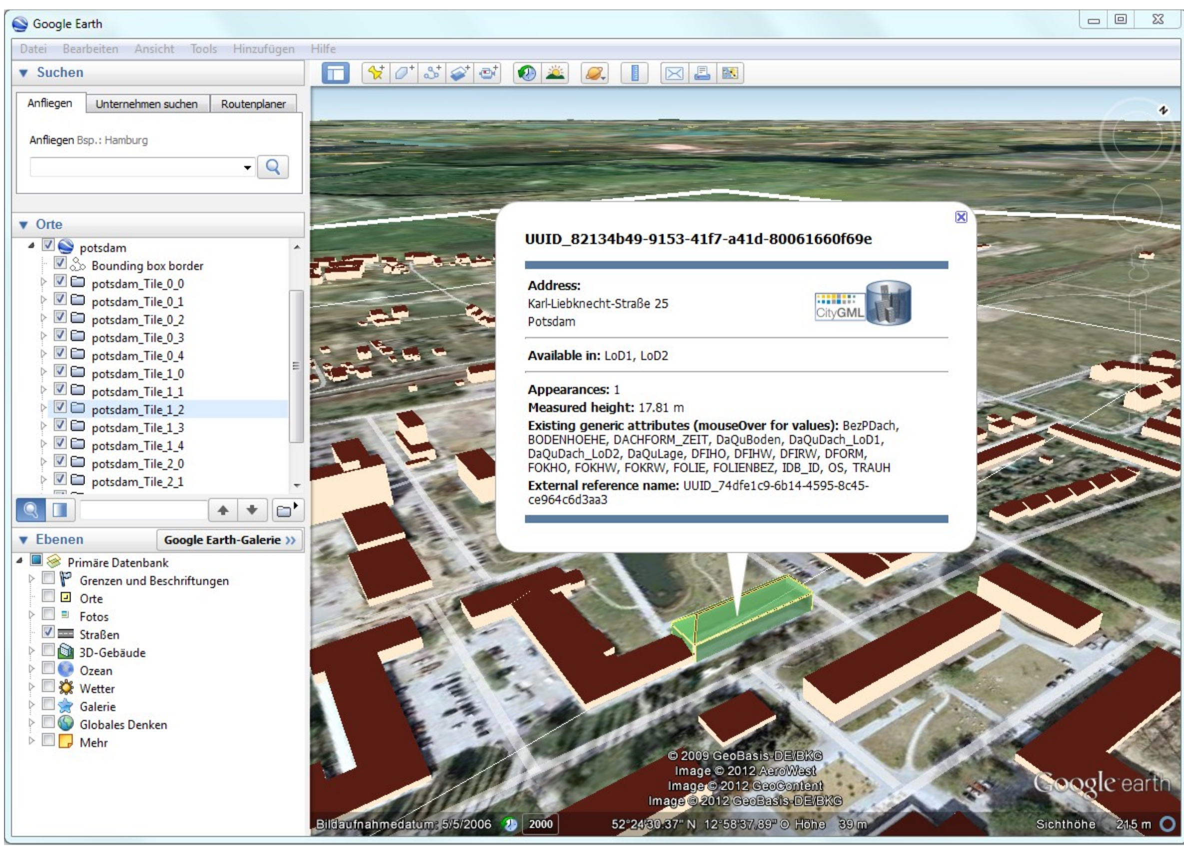


Abb. D11: Das geographische Institut der Universität Potsdam mit Balloon-Template in Google Earth (aus dem 3D-Stadtmodell Potsdam, Eigentum der Stadt Potsdam in Kooperation mit der virtualcitySYSTEMS GmbH)

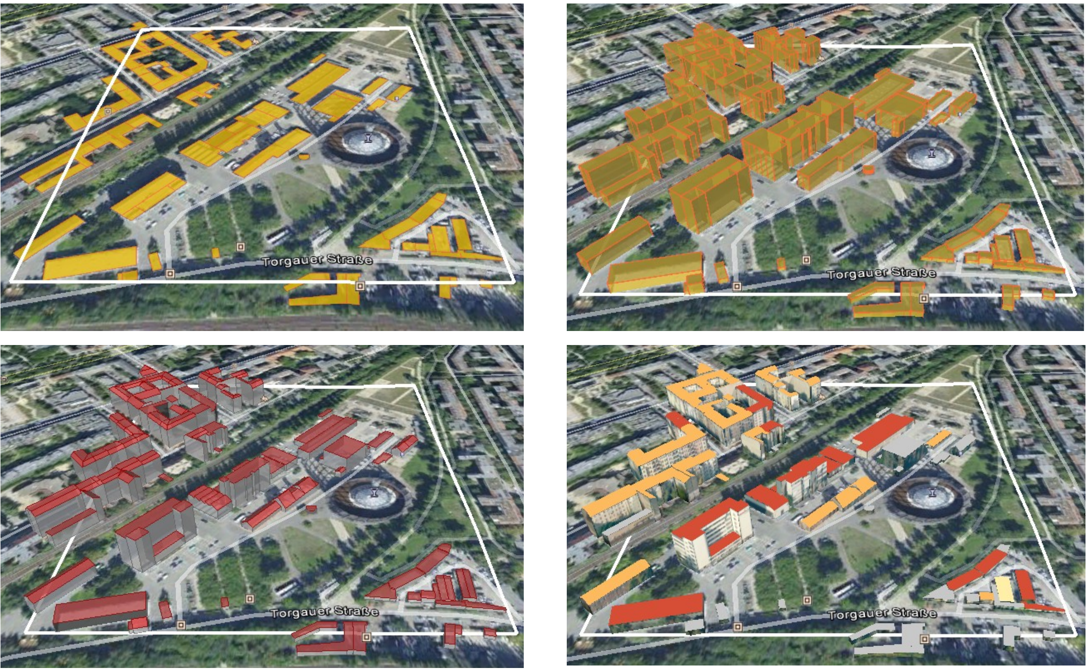
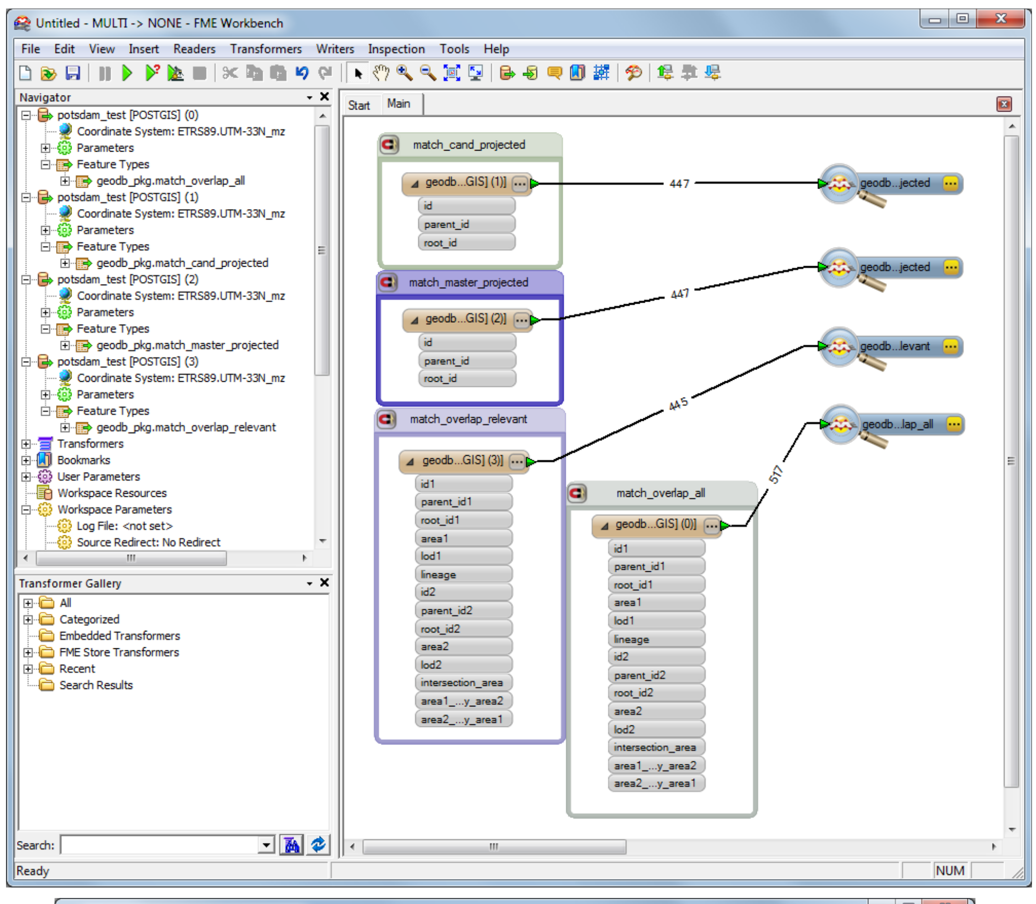


Abb. D12: Die vier Arten des KML-Export (Footprint, Extruded, Geometry, COLLADA) am Beispiel des Gasometer-Datensatzes (Teil des Release)



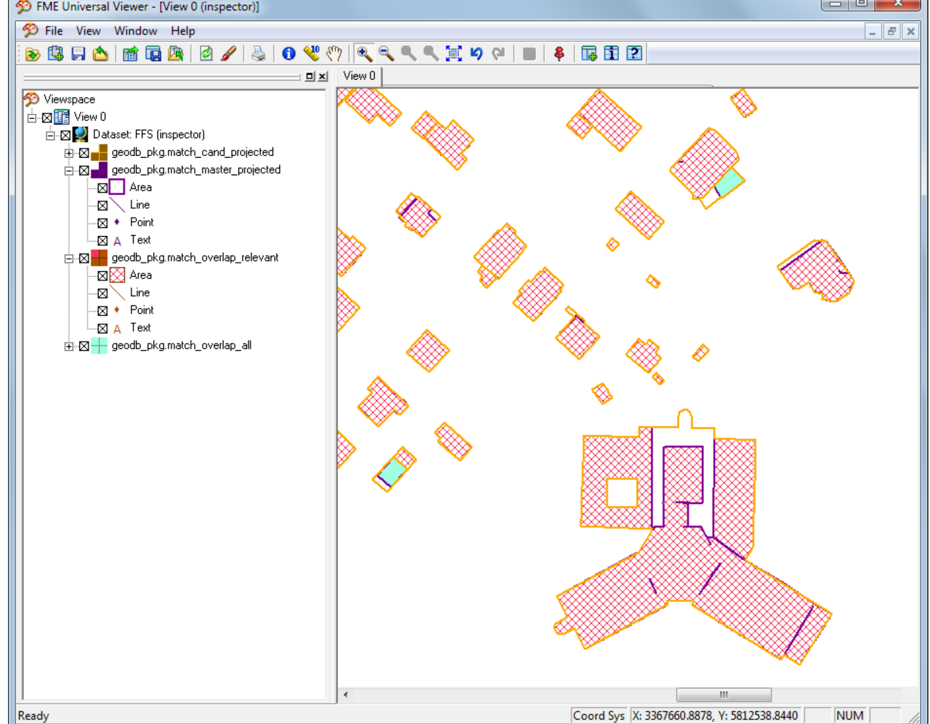


Abb. D13: Visuelle Analyse von Ergebnissen des Matching/Merging-Plugins mit dem FME Inspector (aus dem 3D-Stadtmodell Potsdam, Eigentum der Stadt Potsdam in Kooperation mit der virtualcitySYSTEMS GmbH)

D.3. Sonstige Abbildungen

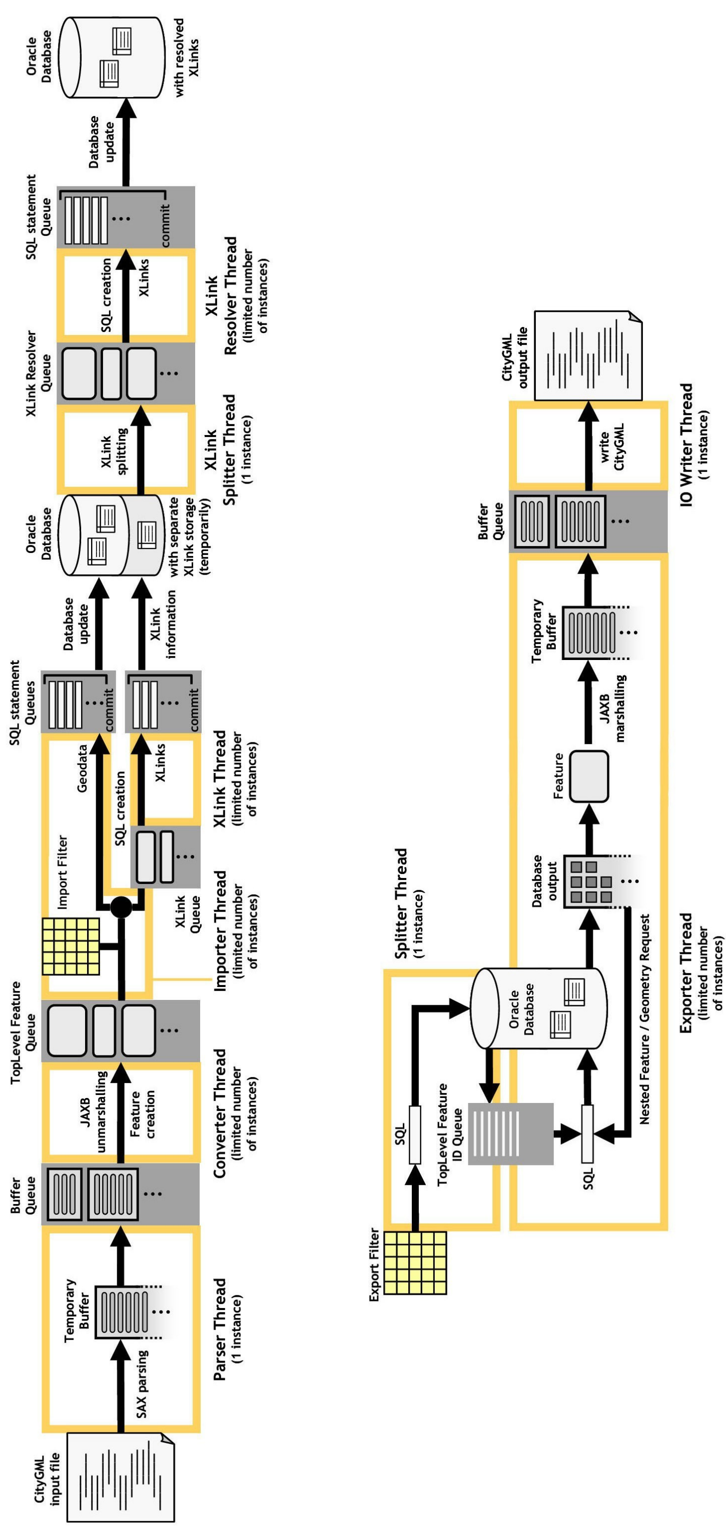


Abb. D14: Prozessketten beim Im- und Export von CityGML-Daten
[Quelle: NAGEL, STADLER 2008]

Abb. D15: Angefertigte Skizze, um einen ersten „Überblick" von der Softwarearchitektur des Importer/Exporter zu erhalten

Anhang E

3D City Database Version 2.0.6-postgis Tutorial

In diesem letzten Anhang befindet sich ein englisches Tutorial zum Aufsetzen der *3DCityDB* in *PostGIS* [vgl. Kunde et al. 2012]. Das Dokument ist Teil der veröffentlichten *PostGIS*-Version und wird mit dem herunterladbaren Setup ausgeliefert. Es kann auch separat von der Projekt-Homepage heruntergeladen werden.

3D City Database for CityGML

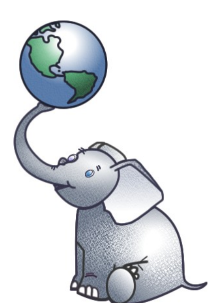
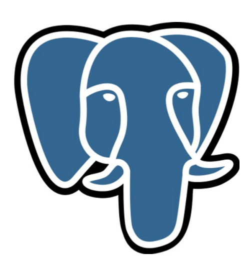
3D City Database Version 2.0.6-postgis

Importer/Exporter Version 1.4.0-postgis

Release Version

Tutorial

27 August 2012

Geoinformation Research Group
Department of Geography
University of Potsdam

Felix Kunde
Hartmut Asche

Institute for Geodesy and
Geoinformation Science
Technische Universität Berlin

Thomas H. Kolbe
Claus Nagel
Javier Herreruela
Gerhard König

(Page intentionally left blank)

Content

Disclaimer.......... **301**

1. Overview.......... **302**
2. Major changes to the Oracle version.......... **305**
3. Requirements.......... **309**
4. How to setup a 3DCityDB in PostGIS.......... **310**
5. FAQ.......... **317**
6. References.......... **319**

Appendix.......... **320**

Disclaimer:

The *3D City Database version 2.0.6-postgis* and the *Importer/Exporter version 1.4.0-postgis* developed by the *Institute for Geodesy and Geoinformation Science (IGG)* at the *Technische Universität Berlin* is free software under the GNU Lesser General Public License Version 3.0. See the file LICENSE shipped together with the software for more details. For a copy of the GNU Lesser General Public License see the files COPYING and COPYING.LESSER [www1].

THE SOFTWARE IS PROVIDED BY *IGG* "AS IS" AND "WITH ALL FAULTS." *IGG* MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE QUALITY, SAFETY OR SUITABILITY OF THE SOFTWARE, EITHER EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

IGG MAKES NO REPRESENTATIONS OR WARRANTIES AS TO THE TRUTH, ACCURACY OR COMPLETENESS OF ANY STATEMENTS, INFORMATION OR MATERIALS CONCERNING THE SOFTWARE THAT IS CONTAINED ON AND WITHIN ANY OF THE WEBSITES OWNED AND OPERATED BY *IGG*.

IN NO EVENT WILL *IGG* BE LIABLE FOR ANY INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES HOWEVER THEY MAY ARISE AND EVEN IF *IGG* HAVE BEEN PREVIOUSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1. Overview

Welcome to the release of the *3D City Database Version 2.0.6-postgis* and the *Importer/Exporter Version 1.4.0-postgis* for *PostGIS*. With the ability to now store and analyze CityGML-documents in *PostGIS*, we are proud to present our free software in a fully OpenSource-context.

Thanks to the continuous development of *PostGIS* 2.0 with new features for topology, raster and 3D support, a long considered port became feasible. Except for version- and history management all key features of the *3D City Database* incl. the *Importer/Exporter* have been translated to *PostgreSQL / PostGIS.* For a quick overview see table 1.

Please note that this document only gives a short introduction on the *PostGIS*-specific details. For a full overview of the *3DCityDB* and the *Importer/Exporter*, please refer to the version 2.0.1 documentation [1] and the addendum [2] for the recent release of the database and the *Importer/Exporter* tool (2.0.6 and 1.4.0).

Tab. 1: Port-overview on supported key-features of both versions

Key Features of the 3D City Database	**Oracle**	**PgSQL**
Semantically rich, hierarchically structured model	✓	✓
Five different Levels of Detail (LODs)	✓	✓
Appearance data in addition to flexible 3D geometries	✓	✓
Representation of generic and prototypical 3D objects	✓	✓
Free, also recursive aggregation of geo objects	✓	✓
Complex digital terrain models (DTMs)	✓	✓
Management of large aerial photographs	✓ (Oracle-specific)	✓ (PostGIS-specific)
Version and history management	✓ (Oracle-specific)	**X**
Matching/merging of building features	✓	✓
Key Features of the Importer/Exporter		
Full support for CityGML 1.0 and 0.4.0	✓	✓
Exports of KML/COLLADA models	✓	✓
Generic KML information balloons	✓	✓
Reading/writing CityGML instance documents of arbitrary file size	✓	✓
Multithreaded programming facilitating high-performance CityGML processing	✓	✓
Resolving of forward and backwards XLinks	✓	✓
XML validation of CityGML documents	✓	✓
User-defined Coordinate Reference Systems	✓	✓
Coordinate transformations for CityGML exports	✓	✓
Matching/merging of building features	✓	✓

✓= equivalent support, ✓= Oracle-specific support ✓= PostGIS-specific support **X** = not supported

3D City Database (abbreviated as *3DCityDB* in the following) [3]

- **Complex thematic modelling:**
  Description of thematic features by attributes, relations, nested aggregation hierarchies (part-whole-relations) between features in a semantic and a geometric manner which is useful for thematic queries, analyses, or simulations

- **Five different Levels of Detail (LODs)**
  Multi-representation of geo objects (including DTMs and aerial photographs) in five different LODs based on geometric precision and thematic refinement

- **Appearance data**
  Appearance of features can be used to represent textures and materials, but also non-visual properties like infra-red radiation, noise pollution, etc.

- **Complex digital terrain models (DTMs)**
  DTMs can be represented in four different ways: by regular grids, triangulated irregular networks (TINs), 3D mass points and 3D break lines. For each LOD a complex relief can be aggregated from any number of DTM components of different types. For example, 3D mass points and break lines can be used together to form complex terrain models.

- **Representation of generic and prototypical 3D objects**
  For efficient memory management, frequently occurring objects at different locations of the city-model can be stored once for each LOD as a prototype and be referred to as an implicit geometry, e.g. pieces of street, furniture like lanterns, road signs, benches etc.

- **Free, also recursive aggregation of geo objects**
  Geo objects can be aggregated to a group according to user-defined criteria. Each group represents a geo object itself. Names and additional classifying attributes can be assigned to groups. Groups may contain other groups as members, resulting in aggregation hierarchies of arbitrary depth.

- **Flexible 3D geometries**
  Geometries of 3D objects can be represented through the combination of surfaces and solids as well as any, also recursive, aggregation of these elements.

- **Management of large aerial photographs**
  The database can handle aerial photographs of arbitrary size using the new raster2pgsql raster-loader of *PostGIS*

Importer/Exporter

- Import and export of even very large CityGML instance documents (> 4 GB)
- Export of buildings into the KML/COLLAD format
- Coordinate transformation and tiling for exports
- Multiple filter-operations for im- and exports incl. a graphical select of a bounding box by a map widget
- Management of user-defined Coordinate Reference Systems (SRIDs)
- Matching and merging of redundant building objects in the database
- New functionalities can be incrementally added via Plugins

Further information, software downloads, ready-to-use demos, links to the source code repository, and much more can be found at:
http://opportunity.bv.tu-berlin.de/software/projects/3dcitydb-imp-exp/ [www2]
and soon at **the official website of the 3D City Database** [www3].

The *PostGIS* port was realized within a Master's thesis by Felix Kunde conducted at the University of Potsdam, and was supported by the *3DCityDB* developer team of the IGG at the Technical University of Berlin as well as the company virtualcitySYSTEMS GmbH (Berlin, Germany). A previous translation of SQL scripts was done by Laure Fraysse in cooperation with IGO (Paris, France) which was the starting point for the further development.

2. Major Changes to the Oracle version

Data modelling and relational schema

The data model behind the relational database schema of the *3DCityDB* was kept unchanged. Supporting UML diagrams and their relational mapping can be found in the main *3DCityDB* documentation [1]. Only two *Oracle*-specific attribute types had to be changed. First, for polygonal geometries the spatial data type SDO_GEOMETRY was replaced by ST_GEOMETRY (see [1]: 18), and second, the ORDImage data type for storing texture images was substituted by a simple BLOB (see [1]: 20).

When referring to the relational schema several differences in data types will always occur when using a different Database Management Systems (DBMS). Their internal structure is mostly following the same purpose, so only the name has to be switched. The following table lists the differences between *Oracle Spatial* and *PostgreSQL / PostGIS*:

Tab. 2: Differences in data types

Oracle Spatial	PostgeSQL/PostGIS	further explanation
varchar2	varchar	
number	numeric	integer used for referential id_columns because of serial
binary_double	double precision	
blob, clob	bytea, text	
	serial (integer)	implicitly creates a sequence named `tablename_columnname_seq`
ORDImage	bytea	PostGIS raster might be an option
sdo_geometry	(st_)geometry	st_geometry also exists in Oracle
sdo_raster	raster	formerly known as WKT Raster
sdo_georaster	raster	

Creating geometric columns and spatial indexes

PostGIS 2.0 introduces the *PostgreSQL* type modifier definition of geometry-columns inside of CREATE TABLE statements [www4]. It was used in the SQL scripts instead of the older but still common function AddGeometryColumn. Unlike the explicit definition for `USER_SDO_GEOM_METADATA` in *Oracle Spatial*, both methods implicitly insert a tuple of metadata in the `geometry_columns`-view.

```sql
CREATE TABLE surface_geometry(
  id          NUMBER NOT NULL,
  geometry    SDO_GEOMETRY,
  . . .
)

INSERT INTO USER_SDO_GEOM_METADATA (TABLE_NAME, COLUMN_NAME, DIMINFO, SRID)
  VALUES ('SURFACE_GEOMETRY', 'GEOMETRY',
    MDSYS.SDO_DIM_ARRAY
      (MDSYS.SDO_DIM_ELEMENT('X', 0.000, 10000000.000, 0.0005),
       MDSYS.SDO_DIM_ELEMENT('Y', 0.000, 10000000.000, 0.0005),
       MDSYS.SDO_DIM_ELEMENT('Z', -1000, 10000, 0.0005)) , 3068);
```

ORACLE DATABASE 11g

```sql
CREATE TABLE surface_geometry(
  id          SERIAL NOT NULL,
  geometry    GEOMETRY(PolygonZ,3068),
  . . .
)
```

PostgreSQL PostGIS

or

```sql
CREATE TABLE surface_geometry(
  id          SERIAL NOT NULL,
  . . .
)

SELECT AddGeometryColumn('surface_geometry', 'geometry', 3068, 'POLYGON', 3);
```

The columns that store 3D-geometries are indexed using an n-dimensional GiST index. It is a common practice to force *PostgreSQL* to get rid of dead rows and update table statistics for the spatial columns after bulk inserts or updates and not wait until the "autovaccum-deamon" of *PostgreSQL* will do that [www5] [4]. This is done by the VACUUM ANALYSE command:

```sql
VACUUM ANALYZE [table_name] [(column_name)];
```

The SQL Planner will use these statistics to evaluate if a GiST index should be used for spatial queries. A SQL script provided to save the user from having to manually write SQL commands for all affected columns. It is shipped within the folder 3dcitydb/postgis/UTIL.

Raster-data management

The raster-data management is much simpler than in *Oracle Spatial*. In *Oracle* a SDO_GeoRaster object must relate to a raster-data-table (RDT) with SDO_Raster objects which hold the actual raster-files. The *Oracle* version of the *3DCityDB* also contains tables for initial imports of image-tiles which can be merged to one raster-file in a second step (IMP-tables). They are necessary for a raster-import tool that was developed for a former version of the *3DCityDB* [5]. *PostGIS 2.0* offers a simple but powerful tool for importing raster-files into the database called *raster2pgsql*. It is executed from the command line and creates a proper SQL insert command for the selected raster-file. Operators can and should be used for setting the reference system, tiling and different levels for raster-overviews (pyramid-layers) (see

example below). In the *PostGIS* approach every stored tile is a raster-object itself, even the raster-overviews. They can be grouped by their original file-name stored in a separate column. This concept would make the RDT and IMP-tables obsolete. Therefore the RDT- and IMP-tables were dropped for the *PostGIS* version of the *3DCityDB*. We recommend to use the *raster2pgsql* tool.

An import into the `raster_relief`-table could look like this:

```
raster2pgsql -f rasterproperty -s 3068 -I -C -F -t 128x128 -l 2,4 relief/*.tif
     raster_relief > rastrelief.sql
```

`-f` sets the name of the target column (in this case `rasterproperty`), the target table is specified at last (`raster_reflief`)
`-s` sets the srid for the raster
`-F` adds a column for the original file name (file.format)
`-t` tiling-operator
`-l` levels for raster-overviews

It is possible to import multiple raster-files from a given folder like in the example (`relief/*.tif`). For further readings please refer to the *PostGIS* documentation on raster data management [www6].

History Management

Based on the *Oracle Workspace Manager* it is possible to manage concurrent versions or planning scenarios of the *3DCityDB* within one user schema. They are organized as views of the original dataset or of their parent version. The Oracle version of the *3DCityDB* delivers scripts to enable or disable versioning support for the database tables as well as a bundle of scripts and tools for managing planning-alternatives called the *Planning Manager.* Unfortunately, as *PostgreSQL* does not offer any equivalent facility, the *Planning Manager* and related scripts could not be ported. Corresponding elements in the graphical user interface (GUI) of the *Importer/Exporter* were removed.

A few free projects exist which implement script-based solutions [www7, www8], but for features like the *Planning Manager* they would need a lot of code-rework to get the same results like with Oracle's *Workspace Manager*. It will be considered for future releases of the *PostGIS* version.

Oracle packages vs. PostgreSQL schemas

The *3DCityDB* provides PL/pgSQL stored procedures which are used by the *Importer/Exporter*-tool. Fortunately *PostgreSQL*'s procedural language of SQL PL/pgSQL comes close to Oracle's PL/SQL grammar which facilitated the porting of scripts. Note that previous self-developed scripts for the *Oracle* version will not with *3DCityDB v2.0.6-postgis*. They have to be translated to PL/pgSQL first in order to work correctly.

For the *Oracle* version the procedures and functions were grouped into packages. However, regarding *PostgreSQL* the package concept only exists in the commercial *Plus Advance Server* by EnterpriseDB. An alternative grouping mechanism for stored procedures that is suggested by the *PostgreSQL* documentation [www9] and which has been implemented, is the usage of schemas. A schema is a separate namespace with own tables, views, sequences, functions etc. The packages from the *Oracle* release are represented in one *PostgreSQL* schema called `geodb_pkg` and not in several schemas for each package (see also figure 2 on page 16). But for a better overview the functions were given name prefixes:

Tab. 3: Function grouping in Oracle and PostgreSQL

former package name	**Prefix**	**Count**	**Source** (PL_pgSQL/GEODB_PKG/)
geodb_delete_by_lineage	del_by_lin_	1	DELETE/DELETE_BY_LINEAGE.sql
geodb_delete	del_	48	DELETE/DELETE.sql
geodb_idx	idx_	16	INDEX/IDX.sql
geodb_match	match_	12	MATCHING/MATCH.sql
geodb_merge	merge_	9	MATCHING/MERGE.sql
geodb_stat	stat_	1	STATISTICS/STAT.sql
geodb_util	util_	9	UTIL/UTIL.sql

3. Requirements

This chapter provides an overview of the minimum requirements for the *3DCityDB* and the *Importer/Exporter* tool. Please carefully review these requirements.

3D City Database

As illustrated in chapter 2 some of the features of *PostGIS 2.0* are used. Thus the SQL scripts would only work with version *2.0* or higher. *PostGIS 2.0* requires *PostgreSQL 8.4* or higher. For 64-bit Windows OS only *9.0* or higher can be used. An empty *3DCityDB* requires 14 MB of disk-space (11 MB *PostGIS* + 3 MB *3DCityDB*).

Importer/Exporter

The *Importer/Exporter* tool can run on any platform providing support for Java 6. It has been successfully tested on (but is not limited to) the following operating systems: Microsoft Windows XP, Vista, 7; Apple Mac OS X 10.6; Ubuntu Linux 9, 10, 11.

Prior to the setup of the *Importer/Exporter* tool, the Java 6 Runtime Environment (JRE version 1.6.0_05 or higher) or Java 7 Runtime Environment (JRE version 1.7.0_03 or higher) must be installed on your system. The necessary installation package can be obtained from [www10].

The *Importer/Exporter* tool is shipped with a universal installer that will guide you through the steps of the setup process. A full installation of the *Importer/Exporter* including documentation and example CityGML files requires approx. 110 MB of hard disk space. Installing only the mandatory application files will use approx. 16 MB of hard disk space. Installation packages can be chosen during the setup process.

The *Importer/Exporter* requires at least 256 MB of main memory. For the import and export of large CityGML respectively KML/COLLADA files, a minimum of 1 GB of main memory is recommended.

4. How to set up a 3DCityDB in PostGIS

1. Installed RDBMS and configuration

Make sure that the *PostgreSQL* server installation is of version 8.4 or higher. For the right settings of the configuration files check the according *PostgreSQL* online documentation [www11]. The *PostGIS* extension must be of version 2.0.0 or higher. It has to be considered that both projects are under continuous development, but it is recommended that only officially released installers should be used.

2. Run the 3DCityDB-Importer-Exporter-1.4-postgis-Setup

The installer setup of the software is mostly self-explaining. The SQL and PL/pgSQL scripts of the *3DCityDB* are grouped in the 3dcitydb folder at the target installation-path. The folder structure is explained shortly with the next table:

Tab. 4: Folder hierarchy of the *3DCityDB* installation package

		3dcitydb/postgis	Explanation
		CREATE_DB.bat	batchfile that calls CREATE_DB.sql (Microsoft Windows family)
		DROP_DB.bat	batchfile that calls DROP_DB.sql (Microsoft Windows family)
		CREATE_DB.sh	shell script that calls CREATE_DB.sql (UNIX/Linux and derivates, MacOS X)
		DROP_DB.sh	shell script that calls DROP_DB.sql (UNIX/Linux and derivates, MacOS X)
		CREATE_DB.sql	calls SQL scripts for setting up the relational schema of the 3DCityDB
		CREATE_GEODB_PKG.sql	creates a separate schema in the database named geodb_pkg with stored procedures, called by CREATE_DB.sql
		DROP_DB.sql	drops all the schema-elements cascadingly, called by DROB_DB.bat / .sh
		SCHEMA	called by CREATE_DB.sql
		CONSTRAINTS	contains a file that sets the referential foreign keys between tables
		INDEXES	contains files for setting sequential (B-Tree) and spatial indexes (GiST)
		TABLES	contains files for creating the database tables
		PL_pgSQL/GEODB_PKG	called by CREATE_GEODB_PKG.sql
		DELETE	contains scripts that help to delete single features from the database. Used by the Matching-Merging-Plugin for the Importer/Exporter.
		INDEX	contains scripts with index-functions. Only used by the Importer/Exporter.
		MATCHING	contains scripts for the Matching-Merging-Plugin.
		STATISTICS	contains a script that generates a database-report for Importer/Exporter
		UTIL	contains several helper-functions, mostly for Importer/Exporter
		UTIL	called by CREATE_DB.sql
		CREATE_DB	contains additional scripts for CREATE_DB.sql
		RO_USER	contains a script that creates a read-only user for the database
		VACUUM	contains a script that collects table statistics (vacuum) for spatial columns

3. CREATE an empty PostGIS-database

Select a user with privileges to create an empty database with *PostGIS* Extension and also access the *PostGIS* features. No violation of rights should occur when working as a superuser. In the end it should look like in figure 1. The *3DCityDB* will be stored in the public schema, which also contains the *PostGIS* elements like functions, view for spatial metadata and a table for reference systems.

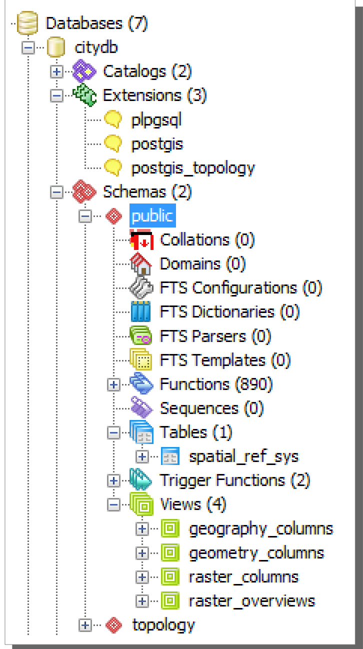


Fig. 1: Empty PostGIS 2.0 database in the pgAdminIII-tool

4. Set up a 3DCityDB

Afterwards, a blank *PostGIS* database is ready to be set up with the relational schema of the *3DCityDB*. The CREATE_DB SQL script in the main folder 3dcitydb/postgis has to be executed from the *psql* console of *PostgreSQL*. It can not be done in the *pgAdminIII* tool. A more comfortable way is offered with shell scripts CREATE_DB.bat for Microsoft Windows family or CREATE_DB.sh for UNIX/Linux and derivates as well as MacOS X. They have to be edited first in order to call the corresponding SQL file (see text box below). The same applies to the shell scripts DROP_DB.bat and DROP_DB.sh.

```
Provide your database details here
PGPORT=5432                 (this is the default PostgreSQL port)
PGHOST=your_host_address    (localhost or server-address)
PGUSER=your_username        (username e.g. postgres)
CITYDB=your_database        (name of the 3D-CityDB e.g. citydb)
PGBIN=path_to_psql.exe      (e.g. 'C:\PostgreSQL\bin' or 'C:\pgAdmin III' on Windows
                             or '/usr/bin' on UNIX/Linux/MacOS X)
```

The Windows batchfiles are then executed by double clicking. The .sh scripts can be run from within a shell environment. Please open your favourite shell and change to the "3dcitydb/postgis" subfolder within the installation directory of the *Importer/Exporter*. Enter the following command to make the CREATE_DB.sh script executable for the owner of the file:

```
chmod u+x CREATE_DB.sh
```

Afterwards, simply run the CREATE_DB.sh script by typing:

```
./ CREATE_DB.sh
```

When executed the user might be asked for his PostgreSQL login password first. The setup requires two mandatory user inputs:

1. Spatial Reference Identifier for newly created geometry objects (SRID),
2. corresponding GML conformant URN encoding for gml:srsName attributes

Make sure to only provide the numeric identifier of the spatial reference system as SRID (e.g., the EPSG code). When prompted for input, the values provided in parentheses are only examples but no default values! The SRID will be checked for its existence in the spatial_ref_sys table of *PostGIS* and if it's appropriate for spatial functions. If the SRID is accepted the user is given the feedback "SRID ok". Otherwise an error will occur which forces the setup to stop.

A successful test session for Berlin will look like this:

```
path_to_your_importer_exporter_installation\resources\3dcitydb\postgis>
"C:\PostgreSQL\bin\psql" -d "citydb" -f "CREATE_DB.sql"
Password:
SET
Please enter a valid SRID (e.g., 3068 for DHDN/Soldner Berlin): 3068
Please enter the corresponding SRSName to be used in GML exports (e.g.
urn:ogc:def:crs,crs:EPSG::3068,crs:EPSG::5783):
urn:ogc:def:crs,crs:EPSG:6.12:3068,crs:EPSG:6.12:5783

CREATE FUNCTION
 check_srid
------------
 SRID ok
(1 row)

CREATE TABLE
ALTER TABLE
INSERT 0 1
CREATE TABLE
ALTER TABLE
...
```

```
...
ALTER TABLE
ALTER TABLE
...
CREATE INDEX
CREATE INDEX
...
INSERT 0 1
INSERT 0 1
...
CREATE SCHEMA
CREATE FUNCTION
CREATE FUNCTION
...
CREATE TYPE
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
DROP TABLE
CREATE TABLE
INSERT 0 1
...
CREATE FUNCTION

3DCityDB creation complete!

path_to_your_importer_exporter_installation\resources\3dcitydb\postgis>
pause
Press any key to continue . . .
```

Error cases – Unknown identifiers of local reference system for the city of Potsdam:

```
path_to_your_importer_exporter_installation\resources\3dcitydb\postgis>
"C:\PostgreSQL\bin\psql" -d "citydb" -f "CREATE_DB.sql"
Password:
SET
Please enter a valid SRID (e.g., 3068 for DHDN/Soldner Berlin): 96734
Please enter the corresponding SRSName to be used in GML exports (e.g.
urn:ogc:def:crs,crs:EPSG::3068,crs:EPSG::5783):
urn:ogc:def:crs:EPSG::325833

CREATE FUNCTION
psql:CREATE_DB.sql:47:  ERROR:  Table  spatial_ref_sys  does  not  contain
the SRID 96734. Insert commands for missing SRIDs can be found at
spatialreference.org

path_to_your_importer_exporter_installation\resources\3dcitydb\postgis>
pause
Press any key to continue . . .
```

After running the CREATE_DB SQL script the *3DCityDB* should look like in figure 2. By the counters you can check if the setup was correct.

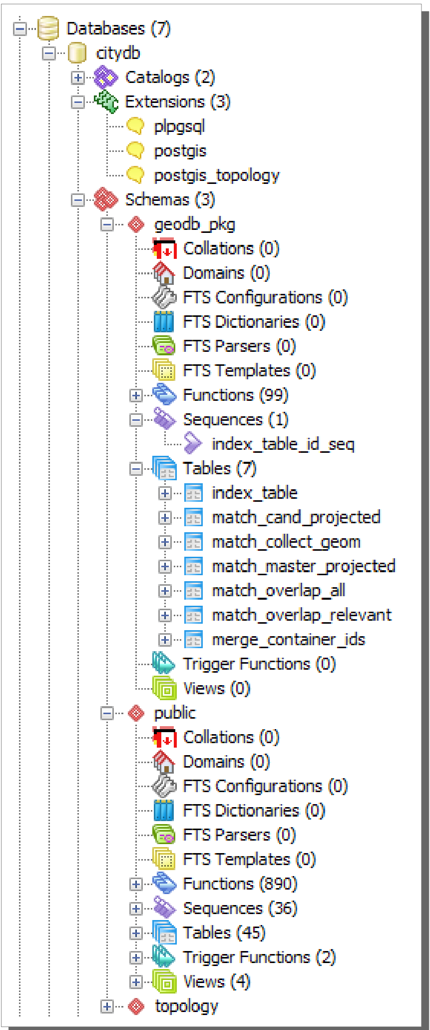


Fig. 2: 3DCityDB in the pgAdminIII-tool

If a wrong reference system was chosen when creating the *3DCityDB* it can still be changed with the function `util_change_db_srid(srid NUMERIC, gml_ident VARCHAR)` found in the GEODB_PKG schema. Of course, the database should still be empty when the SRID is changed to avoid any false projections or errors on the data. It is also possible to reuse an empty *3DCityDB* as a template for any CityGML model to skip the previous steps when creating another database. In case that the SRID is different, just apply the util_change_db_srid function to the new database. The function is executed like this:

```
SELECT geodb_pkg.util_change_db_srid(4326,'urn:ogc:def:crs:EPSG:4326');
```

5. Start the Importer/Exporter

Now that the *3DCityDB* is ready to use, start the *Importer/Exporter* batchfile. After a few seconds the GUI should pop up. Switch to the database-panel and enter your connection details. If you have worked with the *Importer/Exporter* before you will notice that the functionalities are similar to in the *Oracle* version. One difference on the connection details appears at the textfield for the database-name. For connecting to *Oracle* the instance SID had to be entered. For *PostgreSQL / PostGIS* use the name of the database you have created in step 3. If the connection could be established the console-window should look like this:

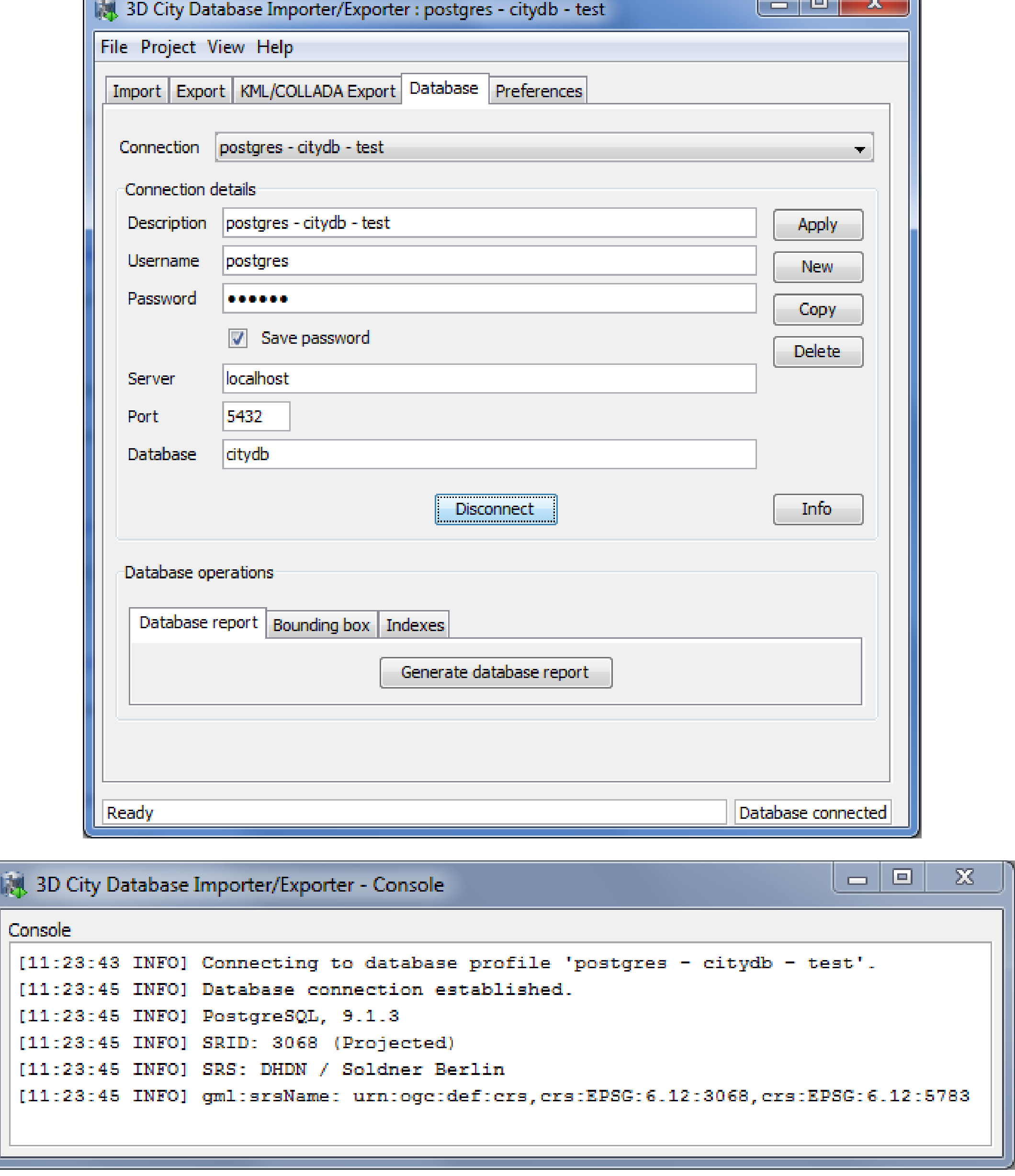


Fig. 3: Importer/Exporter successfully connected to the database (console-window detached)

Again: For further instructions on how to work with the *Importer/Exporter* please read the official *3DCityDB* documentation [1] and the recent addendum [2].

6. DROP the 3DCityDB

To drop an existing database instance of the *3DCityDB* call the SQL script DROP_DB.sql which can be found in the top-level SQL folder of the distribution package. Similar to the setup procedure, the convenience scripts DROP_DB.bat and DROP_DB.sh can be used instead. Please follow the above steps to enter your database details in these scripts and to run them on your machine. Note that DROP_DB.sql only removes the relational schema of the *3DCityDB* as well as all PL/pgSQL functions and utilities. The database itself is not dropped.

5. FAQ

I can not get a connection from the Importer/Exporter to the database. Any help?

Check if you have misspelled some parameter of the connection details. If you are working in a network, check if the *PostgreSQL* configuration file hba.conf contains the address of your client.

Which version of CityGML is supported?

The relational database schema is derived from CityGML version 1.0.0 [6] and is also backwards-compatible with the OGC Best Practices version 0.4.0. The recent release of CityGML 2.0.0 [7] and any Application Domain Extension (ADE) are not yet supported.

How good or bad is the performance of the PostGIS version compared to the Oracle version?

Fair enough. Several im- and exports of CityGML documents of various sizes and contents were tested with both versions. Different city-models were exported to KML/COLLADA, too. In most cases the execution times for the *PostGIS* version reached the same level like the *Oracle* version even with default settings for *PostgreSQL*. It could be noted that untextured CityGML exports were much quicker in *Oracle*. For very large datasets (> 10 GB) *PostgreSQL/PostGIS* scales better for CityGML im- and exports. A detailed analysis comparing the performance of both version is part of the Master's thesis by Felix Kunde [8]. A pdf (german and english) will be available in fall 2012 when the thesis is finished.

Is there a detailed documentation how the port to PostGIS was realized?

A detailed documentation for porting PL/SQL scripts and *Oracle*-specific parts of the Java-code is also shipped with this release. These documents may also provide an introduction for porting own developed features or functions. The Master's thesis will also discuss all aspects on the *PostGIS* port in detail.

Which external tools can I use for visualizing my database?

With the *KML-Exporter* the *Importer/Exporter* took advantage of the widespread *Google Earth* client. You are able to export and watch footprints, extruded footprints, geometries and COLLADA files of your buildings on the virtual globe. You can also switch on a mouse-over highlighting and information balloons for the buildings in the preferences for the *KML-Exporter*.

For CityGML you can use several free viewers e.g. *FZKViewer* or *LandXplorer.* If you want to visualize your data directly from the database you can use the *FME-inspector* in connection with *FME PostGIS-reader*. The next version of the OpenSource-GIS *gvSIG* (1.12) will be able to load 3D models from a *PostGIS* database.
It happens, that external programs with *PostGIS* drivers try to read the geometries with the deprecated *PostGIS* function `asewkb`, which is now called `ST_AsEwkb`. A bundle of lately deprecated functions can be loaded back into your database by executing the legacy.sql file

found in the folder PostgreSQL/share/contrib/postgis-2.0. It can be expected that this mismatch will not appear in recent software-releases

After the KML-Export buildings are flying above the ground. What is going wrong?

Use the default settings in the preferences for the *KML-Exporter* for altitude and terrain. The *KML-Exporter* fetches the heights of Google's elevation-model to calculate the right offset to the buildings in the database. This is also written to the console-window. It is only done once for each building, as the offset is inserted as an generic attribute for the city objects. If you are using CityGML instance documents which were formerly stored in an *Oracle 10g* DBMS and used for KML exports, these entries are holding heights that will not fit *Oracle 11g* or *PostGIS* databases. The coordinate transformation to WGS 84 leads to different height-results between *Oracle 10g* and *11g*. *PostGIS*' `ST_Transform` calculates the same values than *Oracle 11g*. To sum it up: Delete the affected rows in the table `Cityobject_GenericAttrib` (with `attrname` 'GE_LoD[1,2,3 or 4]_zOffset') and restart the KML export. If facing the message `OVER_QUERY_LIMIT` the Limit (2500) for requesting heights from Google's elevation service was exceeded and no values will be written in the database. The user has to wait 24 hours to be able to send new requests to the web service with the same client.

I think I've found a bug ...

If so, please report this bug to us. If you have any further issues on the software performance, results of im- and exports or just questions please tell us. We're glad to receive any feedback. Please note that even though the software was tested thoroughly with various datasets of different size and content (especially when fixing the reported bugs of the beta-release) we cannot guarantee that no more errors occur during imports and exports. It is strongly recommended running the *PostGIS* port in a dedicated testing environment first before managing all your CityGML data with our software.

6. References

Documents:

[1] Kolbe, T.H. ; König, G. ; Nagel, C. ; Stadler, A. (2009): 3D-Geo-Database for CityGML. Version 2.0.1. Documentation. Berlin.
Accessible under: http://www.3dcitydb.net/index.php?id=1897

[2] Kolbe, T.H. ; Nagel, C. ; Herreruela, J. (2012): 3D-Geo-Database for CityGML. Addendum to the 3D City Database Documentation Version 2.0. Berlin.
Accessible under: http://www.3dcitydb.net/index.php?id=1897

[3] Stadler, A. ; Nagel, C. ; König, G. ; Kolbe, T.H. (2009): Making interoperability persistent: A 3D geo database based on CityGML. In: Lee, J. ; Zlatanova, S. (Ed.): 3D Geoinformation Sciences. Lecture Notes in Geoinformation and Cartography. Springer, Berlin / Heidelberg. 175-192.

[4] Obe, R.O. ; Hsu, L. (2010): PostGIS in Action. Manning, New York.

[5] Plümer, l. ; Gröger, G. ; Kolbe, T.H. ; Schmittwilken, J. ; Stroh, V. ; Poth, A. ; Taddeo, U. (2005): 3D Geodatenbank Berlin, Dokumentation V1.0. (in german language only). Institut für Kartographie und Geoinformation der Universität Bonn (IKG), lat/lon GmbH.
Accessible under:
www.businesslocationcenter.de/imperia/md/content/3d/dokumentation_3d_geo_db_berlin.pdf

[6] Gröger, G. ; Kolbe, T.H. ; Czerwinski, A. ; Nagel, C. (2008): OpenGIS City Geography Markup Language (CityGML) Encoding Standard. Version 1.0.0. OGC 08-007rl.
Accessible under: http://www.opengeospatial.org/standards/citygml

[7] Gröger, G. ; Kolbe, T.H. ; Nagel, C. ; Häfele, K-H. (2012): OGC City Geography Markup Language (CityGML) Encoding Standard. Version 2.0.0. OGC 12-019.
Accessible under: http://www.opengeospatial.org/standards/citygml

[8] Kunde, F. (2012): CityGML in PostGIS – Port, usage and performance-analysis using the example of the 3DCityDB of Berlin. Master Thesis. Not yet finished.

Links:

www1 http://www.gnu.org/licenses/
www2 http://opportunity.bv.tu-berlin.de/software/projects/3dcitydb-imp-exp
www3 http://www.3dcitydb.net
www4 http://postgis.refractions.net/docs/AddGeometryColumn.html
www5 http://postgis.refractions.net/documentation/manualsvn/using_postgis_dbmanagement.html#gist_indexes
www6 http://www.postgis.org/documentation/manual-svn/using_raster.xml.html
www7 http://www.kappasys.ch/cms/index.php?id=23
www8 http://pgfoundry.org/projects/temporal/
www9 http://www.postgresql.org/docs/9.1/interactive/plpgsql-porting.html
www10 http://www.java.com/de/download
www11 http://www.postgresql.org/docs/

Appendix:

List of tables and figures:

Table 1: Port-overview on supported key-features of both versions............................302
Table 2: Differences in data types..305
Table 3: Function-grouping with prefixes...308
Table 4: Folder hierarchy of the 3DCityDB installation package...................................310

Figure 1: Empty PostGIS 2.0-database in the pgAdminIII-tool.....................................311
Figure 2: 3DCityDB in the pgAdminIII-tool..314
Figure 3: Importer/Exporter successfully connected to the database.............................315