3D City Database for CityGML

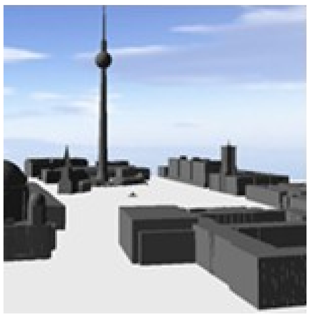
3D City Database Version 2.0.6-postgis

Importer/Exporter Version 1.5.0-postgis

Release Version

Port documentation: Java

21 January 2013



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:

Welcome to the documentation about ported java classes for the *PostGIS* version of the *Importer/Exporter* tool. This document only shows exemplary parts of classes that hold database-specific Java code. Even though they are of a large number the software works mostly database-independent and had not been changed too much in the end. This documentation is divided into thematic parts and not in software packages. Info boxes at the start of each chapter should provide a quick overview which classes had to be changed and which packages were affected by this.

0. Legend

Packages:

☐ api	= no classes in this package were changed
🟨 database	= some parts of this package were changed
🟧 modules	= package contains parts which need to be translated in the future

Location of classes:

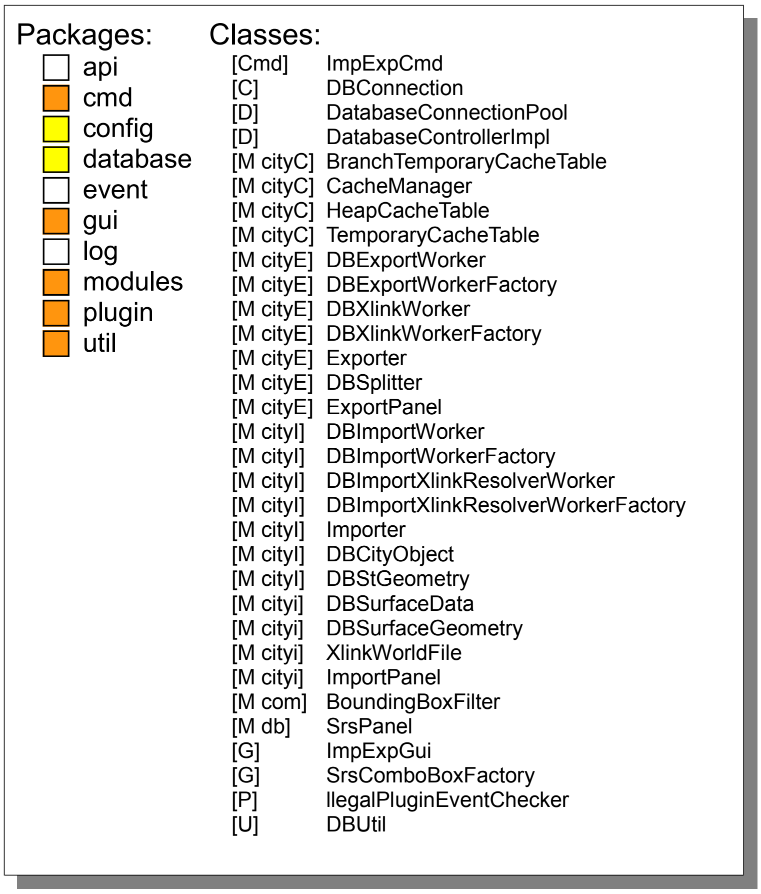
[A]	from package api	[M cityC]	modules.citygml.common
[Cmd]	cmd	[M cityE]	modules.citygml.exporter
[C]	config	[M cityI]	modules.citygml.importer
[D]	database	[M com]	modules.common
[E]	event	[M db]	modules.database
[G]	gui	[M kml]	modules.kml
[L]	log	[M pref]	modules.prefrences
[P]	plugin	[oracle]	oracle.spatial.geometry
[U]	util		

Code:

59	changes start at line 59 in the corresponding class
115+	these lines could not be translated but were also not necessary in function
rep	this code example is repeating itself in the same class
rep+	this code example is repeating itself in the same class and/or in other classes

`//private Integer port = 1521;`	uncommented *Oracle*-specific code (already deleted from the classes)
`private Integer port = 5432;`	*PostGIS*-specific code

1. Connection to the Database



Packages:

☐	api
🟧	cmd
🟨	config
🟨	database
☐	event
🟧	gui
☐	log
🟧	modules
🟧	plugin
🟧	util

Classes:

[Cmd]	ImpExpCmd
[C]	DBConnection
[D]	DatabaseConnectionPool
[D]	DatabaseControllerImpl
[M cityC]	BranchTemporaryCacheTable
[M cityC]	CacheManager
[M cityC]	HeapCacheTable
[M cityC]	TemporaryCacheTable
[M cityE]	DBExportWorker
[M cityE]	DBExportWorkerFactory
[M cityE]	DBXlinkWorker
[M cityE]	DBXlinkWorkerFactory
[M cityE]	Exporter
[M cityE]	DBSplitter
[M cityE]	ExportPanel
[M cityI]	DBImportWorker
[M cityI]	DBImportWorkerFactory
[M cityI]	DBImportXlinkResolverWorker
[M cityI]	DBImportXlinkResolverWorkerFactory
[M cityI]	Importer
[M cityI]	DBCityObject
[M cityI]	DBStGeometry
[M cityi]	DBSurfaceData
[M cityi]	DBSurfaceGeometry
[M cityi]	XlinkWorldFile
[M cityi]	ImportPanel
[M com]	BoundingBoxFilter
[M db]	SrsPanel
[G]	ImpExpGui
[G]	SrsComboBoxFactory
[P]	IllegalPluginEventChecker
[U]	DBUtil

Connection handling has not changed much for the *PostgreSQL* database because the *Universal Connection Pool (UCP)* [www1] by Oracle is still used. The `PoolDataSource` of the *UCP* must pool a proper DataSource of *PostgreSQL*. If using the class `PGSimpleDataSource`, the URL which usually addresses the JDBC driver of an DBMS will not work properly as the result of `conn.getSid()` is not interpreted as the actual database name. To work within a network the server name and the port number would need to be set as well. Therefore the `org.postgresql.Driver` class was chosen in order to be able to use a connection URL. Connection properties were uncommented as the `PGconnection` class of *PostgreSQL* only holds the same attributes than the Java `Connection` class. `CONNECTION_PROPERTY_USE_THREADLOCAL_BUFFER_CACHE` was not offered.

Unfortunately the use of Oracle's *UCP* is not conform to the OpenSource effort behind the *PostGIS* version of the *3DCityDB*. The Apache *Jakarta DBCP* [www2] was tested by the developers but found to work unacceptably worse than the *UCP*. The Connection Pools of Apache's *Tomcat 7* [www3] or *C3PO* [www4] should be an alternative. As seen by the number of orange packages in the overview box changing the connection pooling API would cause a lot of code rework.

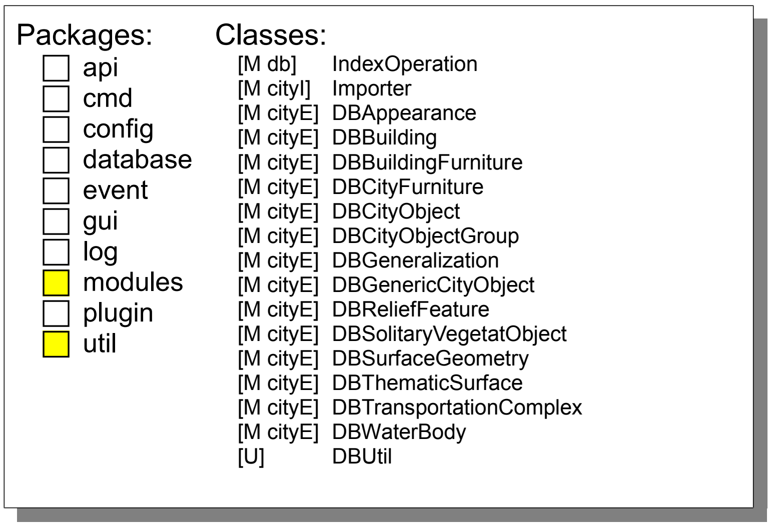
de.tub.citydb.config.project.database.**DBConnection**

```
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());

120+  // set connection properties
```

2. Calling the PL/pgSQL functions



Packages:		Classes:	
☐	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

Most of the functionalities in the database panel of the *Importer/Exporter* are calling stored procedures in the database. So the main changes in code were done in the PL/pgSQL scripts (check the port documentation of PL/SQL scripts for more details) [1]. Within Java only the names of the called functions were changed. The functions are bundled inside of a database schema called "geodb_pkg".

2.1 Index functions, database report, utility functions inside of statements

The bigger the size of files to be imported the longer it takes to index the data after every inserted tuple. It is recommended to drop the indexes before importing big data sets and recreate them afterwards. *Oracle* keeps metadata of a dropped index, *PostgreSQL* does not. An alternative way was programmed but it is not used now. The idea was to just set the index status to invalid (pg_index.indisvalid) that it stays inactive during the import and then REINDEX it afterwards. It was only tested with small datasets but no performance improvement could be detected. The functions are already written but they are not a part of the recent release.

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**

```
64     // private static OracleCallableStatement callableStmt;
       private static CallableStatement callableStmt;

80     // 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
       //    geodb_stat.table_contents}");
rep    // callableStmt.registerOutParameter(1, OracleTypes.ARRAY, "STRARRAY");
       // callableStmt.executeUpdate();
       // ARRAY result = callableStmt.getARRAY(1);
       callableStmt = (CallableStatement)conn.prepareCall("{? = call
             geodb_pkg.stat_table_contents()}");
       callableStmt.registerOutParameter(1, Types.ARRAY);
       callableStmt.executeUpdate();
       Array result = callableStmt.getArray(1);

375    // String call = type == DBIndexType.SPATIAL ?
rep    //          "{? = call geodb_idx.drop_spatial_indexes}" :
       //                "{? = call geodb_idx.drop_normal_indexes}";
```

Drop case:

```
String call = type == DBIndexType.SPATIAL ?
      "{? = call geodb_pkg.idx_drop_spatial_indexes()}" :
            "{? = call geodb_pkg.idx_drop_normal_indexes()}";
```

or Switch case:

```
String call = type == DBIndexType.SPATIAL ?
      "{? = call geodb_pkg.idx_switch_off_spatial_indexes()}" :
            "{? = call geodb_pkg.idx_switch_off_normal_indexes()}";
// callableStmt = (OracleCallableStatement)conn.prepareCall(call);
callableStmt = (CallableStatement)conn.prepareCall(call);
```

2.2 Calculation of the BoundingBox

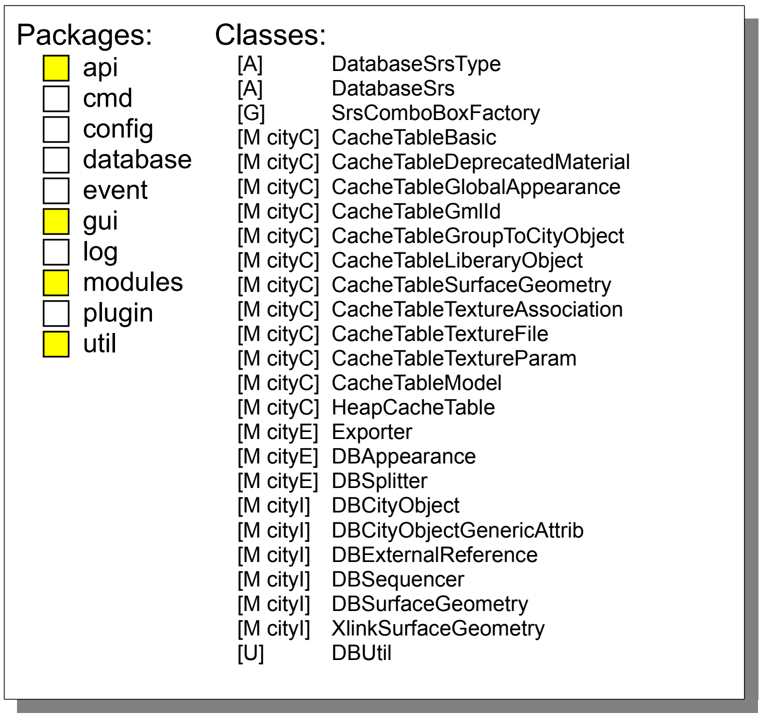
For calculating the BoundingBox workspace variables were uncommented. The query strings had to call equivalent *PostGIS* functions (e.g. `sdo_aggr_mbr --> ST_Extent`, `geodb_util.to2d --> ST_Force_2d`). As rectangle geometries can not be shorten in number of points like in *Oracle* (LLB, URT), 5 Points were needed for the coordinate transformation. As placeholders for single coordinates did not work with a `PreparedStatement` the whole String in the *PostGIS* function `ST_GeomFromEWKT(?)` was used as the exchangeable variable.

de.tub.citydb.util.database.**DBUtil**

```
236   // 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);

629   // 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);

// 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);
```

3. Database specifics in Java



Packages:	
🟨	api
☐	cmd
☐	config
☐	database
☐	event
🟨	gui
☐	log
🟨	modules
☐	plugin
🟨	util

Classes:	
[A]	DatabaseSrsType
[A]	DatabaseSrs
[G]	SrsComboBoxFactory
[M cityC]	CacheTableBasic
[M cityC]	CacheTableDeprecatedMaterial
[M cityC]	CacheTableGlobalAppearance
[M cityC]	CacheTableGmlId
[M cityC]	CacheTableGroupToCityObject
[M cityC]	CacheTableLiberaryObject
[M cityC]	CacheTableSurfaceGeometry
[M cityC]	CacheTableTextureAssociation
[M cityC]	CacheTableTextureFile
[M cityC]	CacheTableTextureParam
[M cityC]	CacheTableModel
[M cityC]	HeapCacheTable
[M cityE]	Exporter
[M cityE]	DBAppearance
[M cityE]	DBSplitter
[M cityI]	DBCityObject
[M cityI]	DBCityObjectGenericAttrib
[M cityI]	DBExternalReference
[M cityI]	DBSequencer
[M cityI]	DBSurfaceGeometry
[M cityI]	XlinkSurfaceGeometry
[U]	DBUtil

3.1 The database SRS

Until now *PostGIS* does not offer 3D spatial reference systems by default. INSERT examples for *PostGIS* can be found at spatialreference.org. Unfortunately 2D and 3D geographic reference systems are equally classified as `GEOGCS`. The function `is3D()` in the `DBUtil` class would not detect 3D SRIDs though. If the INSERT statement by spatialreference.org [www5] is changed manually from `GEOGCS` to `GEOGCS3D is3D()`, it would work because the type is listed in the `DatabaseSrsType` class. It is not sure how 3D SRIDs will be handled in future *PostGIS* releases. *Oracle Spatial* has got some strict rules how to work with them. This includes certain checks on the data, which are not needed for the *PostGIS* version at the moment. It can be noticed that the `spatial_ref_sys` table in *PostGIS* contains less columns than *Oracle*'s `SDO_COORD_REF_SYS` table. Most of the information is stored in the text column `srtext`. It can be extracted with String functions.

de.tub.citydb.api.database.**DatabaseSrsType**

```
//    PROJECTED("Projected"),
//    GEOGRAPHIC2D("Geographic2D"),
//    GEOCENTRIC("Geocentric"),
//    VERTICAL("Vertical"),
//    ENGINEERING("Engineering"),
//    COMPOUND("Compound"),
//    GEOGENTRIC("Geogentric"),
//    GEOGRAPHIC3D("Geographic3D"),
//    UNKNOWN("n/a");
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))");
```

3.2 BoundingBox filter and OptimizerHints in DBSplitter.java

`DBSplitter.java` manages the filtering of data by a given bounding box. In *Oracle Spatial* the spatial operation SDO_RELATE is used for that. SDO_RELATE checks topological relations between geometries according to the 9-intersection Matrix (DE-9IM). It is possible to combine similar mask attributes with a logical OR (+) like 'inside' and 'coveredby'. More attributes should stand in their own SDO_RELATE statements, all concatenated using UNION ALL.

The equivalent *PostGIS* function is ST_Relate. There are some slight differences but before digressing into details it should be said that ST_Relate is not using the GiST index of *PostgreSQL*. Therefore the query would be much slower than in the *Oracle* version. In *PostGIS* topological relations are usually queried using functions that are called like the mask attributes like ST_CoveredBy, ST_Inside, ST_Equal etc. Those operations are using the spatial index and work much faster.

Another feature of *Oracle* which is used in the `DBSplitter` class is the “Optimizer Hint”. It is used to tell the internal query optimizer which query plan to prefer. As there are no such Optimizer Hints in *PostgreSQL* they were uncommented.

de.tub.citydb.modules.citygml.exporter.database.content.**DBSplitter**

```
//    bboxFilter = new String[overlap ? 3 : 2];
//
//    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'";
bboxFilter = new String[overlap ? 2 : 1];

String geomAgeomB = "(co.ENVELOPE, " +
      "ST_GeomFromEWKT('SRID=" + dbSrs.getSrid() + ";POLYGON((" +
            minX + " " + minY + "," +
            minX + " " + maxY + "," +
            maxX + " " + maxY + "," +
            maxX + " " + minY + "," +
            minX + " " + minY + "))'))";

bboxFilter[0] = "ST_CoveredBy" + geomAgeomB + " = 'TRUE'";

if (overlap)
      bboxFilter[0] = "ST_Intersects" + geomAgeomB + " = 'TRUE'";
```

3.3 Queries for the Import

Some queries in the java classes use database-specific functions which had to be changed.

de.tub.citydb.modules.citygml.exporter.database.content.**DBAppearance**

```
130   // nvl(sd.TEX_IMAGE.getContentLength(), 0) as DB_TEX_IMAGE_SIZE,
rep   // 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,
```

de.tub.citydb.modules.citygml.importer.database.content.**DBCityObject**

```
133   // SYSDATE
      now()
```

de.tub.citydb.modules.citygml.importer.database.content.**DBCityObjectGenericAttrib**

```
64    // 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**

```
91    // 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");

100   // SURFACE_GEOMETRY_SEQ.nextval
      nextval('SURFACE_GEOMETRY_ID_SEQ')
```

3.4 Create Table without “nologging“

A `nologging` option for CREATE TABLE statements has only been offered since *PostgreSQL 9.1* (UNLOGGED parameter). In order to provide backwards compatibility until *PostgreSQL 8.4* the option is not used in the *PostGIS* version.

de.tub.citydb.modules.citygml.common.database.cache.model.**CacheTableModel**

```
// " nologging" +
```

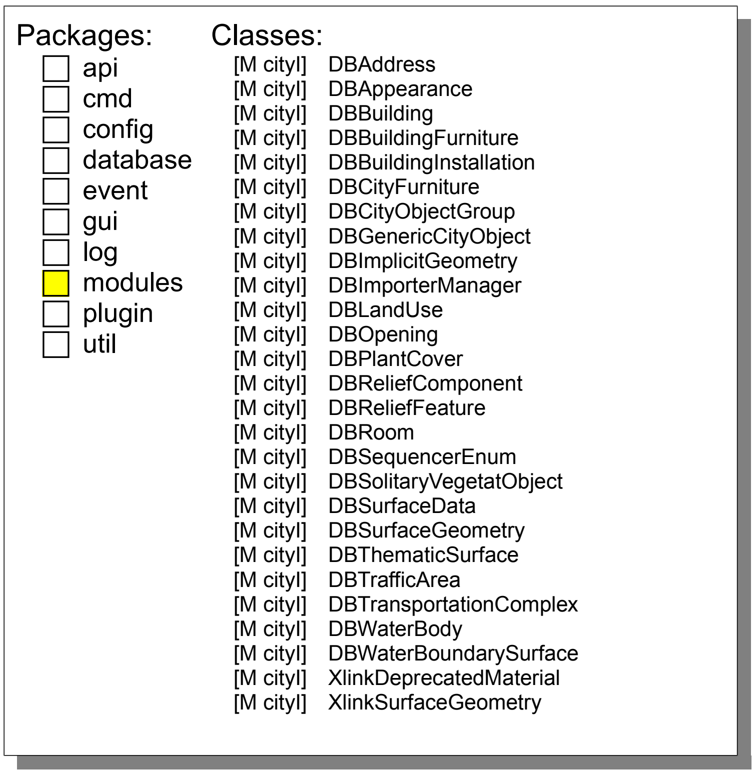
de.tub.citydb.modules.citygml.common.database.cache.**HeapCacheTable**

```
model.createIndexes(conn, tableName/*, "nologging"*/);
```

3.5 Data types in cached tables

In the folder common.database.cache.model several classes had to be changed due to different data types of the DMBS. NUMBER to NUMERIC (ID columns = INTEGER), VARCHAR2 to VARCHAR.

4. Implicit sequences



Packages:
- ☐ api
- ☐ cmd
- ☐ config
- ☐ database
- ☐ event
- ☐ gui
- ☐ log
- 🟨 modules
- ☐ plugin
- ☐ util

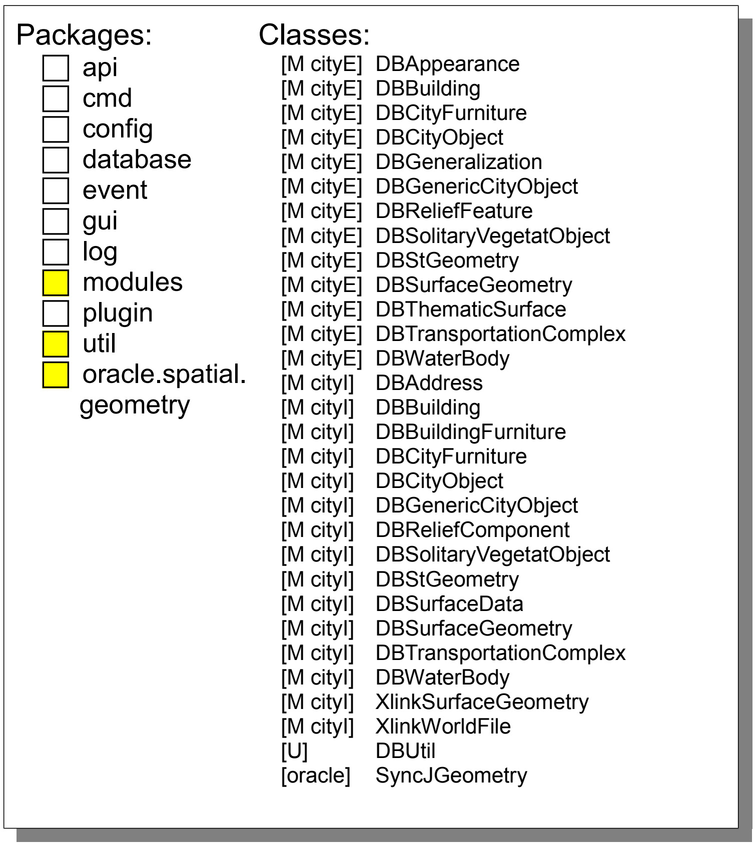
Classes:
[M cityl] DBAddress
[M cityl] DBAppearance
[M cityl] DBBuilding
[M cityl] DBBuildingFurniture
[M cityl] DBBuildingInstallation
[M cityl] DBCityFurniture
[M cityl] DBCityObjectGroup
[M cityl] DBGenericCityObject
[M cityl] DBImplicitGeometry
[M cityl] DBImporterManager
[M cityl] DBLandUse
[M cityl] DBOpening
[M cityl] DBPlantCover
[M cityl] DBReliefComponent
[M cityl] DBReliefFeature
[M cityl] DBRoom
[M cityl] DBSequencerEnum
[M cityl] DBSolitaryVegetatObject
[M cityl] DBSurfaceData
[M cityl] DBSurfaceGeometry
[M cityl] DBThematicSurface
[M cityl] DBTrafficArea
[M cityl] DBTransportationComplex
[M cityl] DBWaterBody
[M cityl] DBWaterBoundarySurface
[M cityl] XlinkDeprecatedMaterial
[M cityl] XlinkSurfaceGeometry

In *PostgreSQL* it is a common practice to assign the data type SERIAL to ID columns which are used as primary keys. SERIAL implicitly creates a sequence with the names of table, column and the ending "_SEQ". The declaration "CREATE SEQUENCE" must not be written manually like in *Oracle*. But this holds a trap. As names are created automatically with SERIAL they differ from the customized names in *Oracle*. See also ***3.3*** for examples.

de.tub.citydb.modules.citygml.importer.database.content.**DBSequencerEnum**

```
//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,
}
```

5. How to work with database geometries in Java



Packages:

- ☐ api
- ☐ cmd
- ☐ config
- ☐ database
- ☐ event
- ☐ gui
- ☐ log
- 🟨 modules
- ☐ plugin
- 🟨 util
- 🟨 oracle.spatial.geometry

Classes:

[M cityE]	DBAppearance
[M cityE]	DBBuilding
[M cityE]	DBCityFurniture
[M cityE]	DBCityObject
[M cityE]	DBGeneralization
[M cityE]	DBGenericCityObject
[M cityE]	DBReliefFeature
[M cityE]	DBSolitaryVegetatObject
[M cityE]	DBStGeometry
[M cityE]	DBSurfaceGeometry
[M cityE]	DBThematicSurface
[M cityE]	DBTransportationComplex
[M cityE]	DBWaterBody
[M cityI]	DBAddress
[M cityI]	DBBuilding
[M cityI]	DBBuildingFurniture
[M cityI]	DBCityFurniture
[M cityI]	DBCityObject
[M cityI]	DBGenericCityObject
[M cityI]	DBReliefComponent
[M cityI]	DBSolitaryVegetatObject
[M cityI]	DBStGeometry
[M cityI]	DBSurfaceData
[M cityI]	DBSurfaceGeometry
[M cityI]	DBTransportationComplex
[M cityI]	DBWaterBody
[M cityI]	XlinkSurfaceGeometry
[M cityI]	XlinkWorldFile
[U]	DBUtil
[oracle]	SyncJGeometry

Translating the processing of geometries to the *PostGIS* JDBC driver was with no doubt the toughest job to do. This chapter shortly explains how geometries were parsed from a CityGML document and inserted into the database and all the way back.

5.1 From CityGML to 3DCityDB

The *Oracle* JDBC driver handles geometries with one central class called `JGeometry`. One instance of `JGeometry` represents SDO_GEOMETRY in the Java world. All methods of different geometric types return `JGeometry`. They need three variables: an array of coordinates, the number of dimensions and a known SRID. The geometries of CityGML are described by geometric primitives from the *citygml4j* library. Their values are first transferred to list elements and then iterated into arrays to be used by the described `JGeometry` methods. `JGeometry` can not be set as an object for the database statements. It needs to be "stored" into a `STRUCT` object, which is a wrapper class for `JGeometry`. This wrapper makes the object more generic to be used by the `PreparedStatement` method `setObject`.

For the *PostGIS* JDBC driver the combination of geometry class and wrapper class is represented by `Geometry` and `PGgeometry`. `Geometry` offers some geometric operations, but to create an instance of `Geometry` the `PGgeometry` method `geomFromString(String)` has to be used. The values of the list elements have to iteratively build up a string and not fill an array. The String represents the geometries in Well Known Text (WKT), which means blank spaces between coordinates (x y z) instead of commas. To be interpreted by the database the geometries have to be wrapped as a `PGgeometry` object and then set for the `PreparedStatement`. See the following figures for a better understanding (the red arrows stand for iterations):

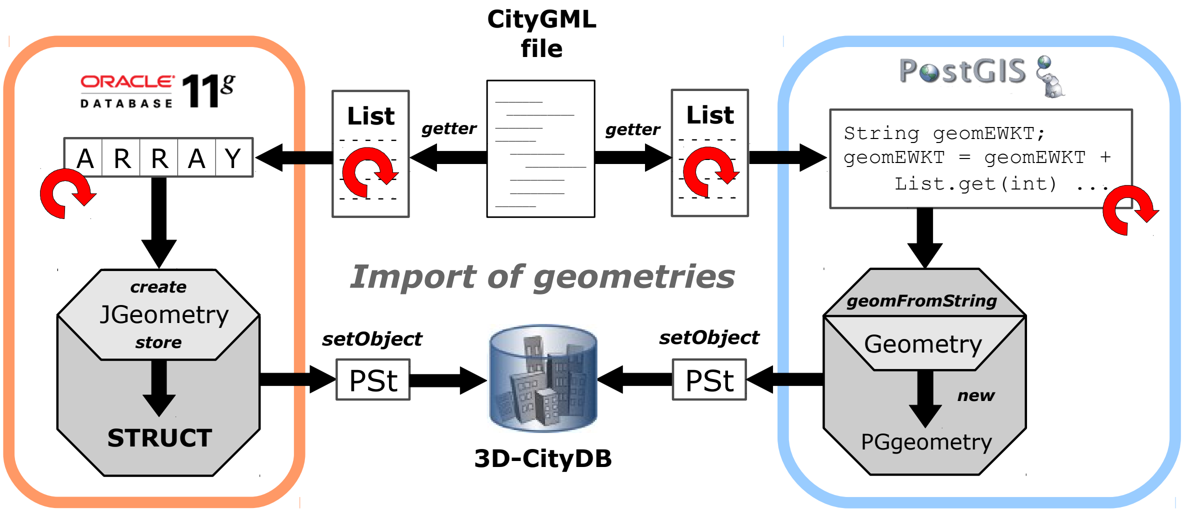


Fig. 1: Import of geometries in Java (PSt = PreparedStatement) (Kunde 2012 [2])

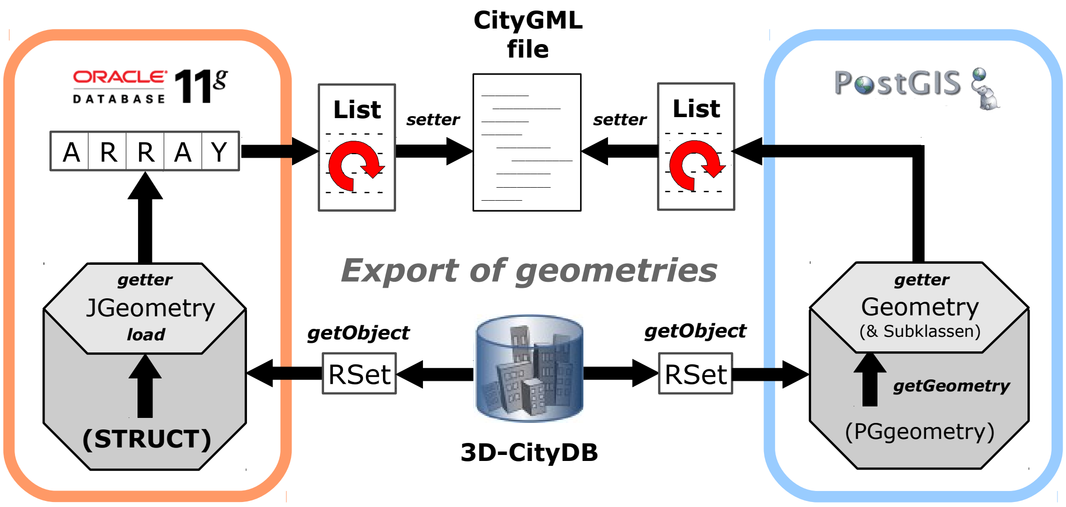


Fig. 2: Export of geometries in Java (RSet = ResultSet) (Kunde 2012 [2])

de.tub.citydb.modules.citygml.importer.database.content.**DBAddress**

```
94      // private DBSdoGeometry sdoGeometry;
rep+    private DBStGeometry stGeometry;

114     // sdoGeometry = (DBSdoGeometry)dbImporterManager.getDBImporter(
rep+          DBImporterEnum.SDO_GEOMETRY);
        stGeometry = (DBStGeometry)dbImporterManager.getDBImporter(
              DBImporterEnum.ST_GEOMETRY);

138     // JGeometry multiPoint = null;
rep+    PGgeometry multiPoint = null;

260     // multiPoint = sdoGeometry.getMultiPoint(address.getMultiPoint());
rep+    multiPoint = stGeometry.getMultiPoint(address.getMultiPoint());

274     // if (multiPoint != null) {
rep+    //    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**

```
214     // double[] ordinates = new double[points.size()];
rep+    // 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.**DBImporterEnum**

```
68      // SDO_GEOMETRY();
        ST_GEOMETRY();
```

de.tub.citydb.modules.citygml.importer.database.content.**DBStGeometry**

```
88    //    public JGeometry getPoint(Point point) {
rep   //          JGeometry pointGeom = null;
      //
      //          if (point != null) {
      //                List<Double> values = point.toList3d();
      //                if (values != null && !values.isEmpty())
      //                      pointGeom = JGeometry.createPoint(toArray(values),
      //                                        3, dbSrid);
      //          }
      //          return pointGeom;
      //    }
      public PGgeometry getPoint(Point point) throws SQLException {
            PGgeometry pointGeom = null;

            if (point != null) {
                  List<Double> values = point.toList3d();

                  if (values != null && !values.isEmpty()) {
                        if (affineTransformation)
                              dbImporterManager.getAffineTransformer().
                                    transformCoordinates(values);

                        pointGeom = new PGgeometry(PGgeometry.geomFromString(
                                          "SRID=" + dbSrid + ";POINT(" +
                                          values.get(0) + " " + values.get(1) +
                                          " " + values.get(2) + ")"));
                  }
            }
            return pointGeom;
      }

141   //    public JGeometry getMultiPoint(MultiPoint multiPoint) {
rep   //          JGeometry multiPointGeom = null;
      //
      //          if (multiPoint != null) {
      //           List<List<Double>> pointList = new ArrayList<List<Double>>();
      //
      //           if (multiPoint.isSetPointMember()) {
      //            for (PointProperty pointProperty :
      //                multiPoint.getPointMember())
      //
      //                if (pointProperty.isSetPoint())
      //                   pointList.add(pointProperty.getPoint().toList3d());
      //
      //                } else if (multiPoint.isSetPointMembers()) {
      //                      PointArrayProperty pointArrayProperty =
      //                            multiPoint.getPointMembers();
      //                      for (Point point : pointArrayProperty.getPoint())
      //                            pointList.add(point.toList3d());
      //                }
      //
      //                if (!pointList.isEmpty())
      //                      multiPointGeom = JGeometry.createMultiPoint(
      //                                  toObjectArray(pointList), 3, dbSrid);
      //          }
      //          return multiPointGeom;
      //    }
      public PGgeometry getMultiPoint(MultiPoint multiPoint) throws SQLException
      {
            PGgeometry multiPointGeom = null;

            if (multiPoint != null); {
                  List<List<Double>> pointList = new ArrayList<List<Double>>();

                  if (multiPoint.isSetPointMember()) {
                        for (PointProperty pointProperty :
                                          multiPoint.getPointMember())
                              if (pointProperty.isSetPoint())
                                    pointList.add(
                                          pointProperty.getPoint().toList3d());

                  } else if (multiPoint.isSetPointMembers()) {
                        PointArrayProperty pointArrayProperty =
                              multiPoint.getPointMembers();
                        for (Point point : pointArrayProperty.getPoint())
                              pointList.add(point.toList3d());
                  }

                  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 = new PGgeometry(
                                          PGgeometry.geomFromString(geomEWKT));
                  }
            }
            return multiPointGeom;
      }
```

223 rep
```
//    public JGeometry getCurve(AbstractCurve curve) {
//          JGeometry curveGeom = null;
//
//          if (curve != null) {
//                List<Double> pointList = new ArrayList<Double>();
//                generatePointList(curve, pointList, false);
//                if (!pointList.isEmpty())
//                      curveGeom = JGeometry.createLinearLineString(
//                            toArray(pointList), 3, dbSrid);
//          }
//          return curveGeom;
//    }
```

```
public PGgeometry getCurve(AbstractCurve curve) throws SQLException {
      PGgeometry curveGeom = null;

      if (curve != null) {
            List<Double> pointList = new ArrayList<Double>();
            generatePointList(curve, pointList, false);
            if (!pointList.isEmpty()) {
                  String geomEWKT = "SRID=" + dbSrid + ";LINESTRING(";

                  for (int i=0; i<pointList.size(); i+=3){
                        geomEWKT += pointList.get(i) + " " +
                                    pointList.get(i+1) + " " +
                                    pointList.get(i+2) + ",";
                  }
                  geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
                  geomEWKT += ")";
                  curveGeom = new PGgeometry(
                                    PGgeometry.geomFromString(geomEWKT));
            }
      }
      return curveGeom;
}
```

```
243   //    public JGeometry getMultiCurve(MultiCurve multiCurve) {
rep   //          JGeometry multiCurveGeom = null;
      //
      //          if (multiCurve != null) {
      //           List<List<Double>> pointList = new ArrayList<List<Double>>();
      //
      //           if (multiCurve.isSetCurveMember()) {
      //                for (CurveProperty curveProperty :
      //                            multiCurve.getCurveMember()) {
      //                      if (curveProperty.isSetCurve()) {
      //                         AbstractCurve curve = curveProperty.getCurve();
      //                         List<Double> points = new ArrayList<Double>();
      //                         generatePointList(curve, points, false);
      //
      //                         if (!points.isEmpty())
      //                            pointList.add(points);
      //                      }
      //                }
      //           } else if (multiCurve.isSetCurveMembers()) {
      //                CurveArrayProperty curveArrayProperty =
      //                                        multiCurve.getCurveMembers();

      //                for (AbstractCurve curve :
      //                            curveArrayProperty.getCurve()) {
      //                      List<Double> points = new ArrayList<Double>();
      //                      generatePointList(curve, points, false);
      //
      //                      if (!points.isEmpty())
      //                            pointList.add(points);
      //                }
      //          }
      //
      //          if (!pointList.isEmpty())
      //                multiCurveGeom = JGeometry.createLinearMultiLineString(
      //                                  toObjectArray(pointList), 3, dbSrid);
      //            }
      //            return multiCurveGeom;
      //      }
      public PGgeometry getMultiCurve(MultiCurve multiCurve) throws SQLException
      {
         PGgeometry multiCurveGeom = null;

            if (multiCurve != null) {
               List<List<Double>> pointList = new ArrayList<List<Double>>();

               if (multiCurve.isSetCurveMember()) {

               for (CurveProperty curveProperty  : multiCurve.getCurveMember()){
                   if (curveProperty.isSetCurve()) {
                        AbstractCurve curve = curveProperty.getCurve();
                        List<Double> points = new ArrayList<Double>();
                        generatePointList(curve, points, false);

                        if (!points.isEmpty())
                           pointList.add(points);
                        }
                   }
               } else if (multiCurve.isSetCurveMembers()) {
                   CurveArrayProperty curveArrayProperty =
                                                   multiCurve.getCurveMembers();
                   for (AbstractCurve curve : curveArrayProperty.getCurve()) {
                       List<Double> points = new ArrayList<Double>();
                       generatePointList(curve, points, false);

                       if (!points.isEmpty())
                           pointList.add(points);
                       }
               }

               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 = new PGgeometry(
                                       PGgeometry.geomFromString(geomEWKT));
                  }
            }
            return multiCurveGeom;
        }

389     //      public JGeometry getPoint(PointProperty pointProperty) {
rep     //              return pointProperty != null ?
        //                      getPoint(pointProperty.getPoint()) : null;
        //      }
        public PGgeometry getPoint(PointProperty pointProperty) throws
                        SQLException {
                return pointProperty != null ?
                        getPoint(pointProperty.getPoint()) : null;
        }

709     //      if (!pointList.isEmpty()) {
        //              Object[] pointArray = new Object[pointList.size()];
        //              int dim = is2d ? 2 : 3;
        //
        //              // if we have to return a 2d polygon we first have to correct
        //              // the double lists we retrieved from citygml4j as they are
        //              // always 3d
        //              if (is2d) {
        //                      for (List<Double> coordsList : pointList) {
        //                              Iterator<Double> iter = coordsList.iterator();
        //                              int count = 0;
        //                              while (iter.hasNext()) {
        //                                      iter.next();
        //
        //                                      if (count++ == 2) {
        //                                              count = 0;
        //                                              iter.remove();
        //                                      }
        //                              }
        //                      }
        //              }
        //
        //              int i = 0;
        //              for (List<Double> coordsList : pointList) {
        //                      double[] coords = new double[coordsList.size()];
        //
        //                      int j = 0;
        //                      for (Double coord : coordsList)
        //                              coords[j++] = coord.doubleValue();
        //                      pointArray[i++] = coords;
        //              }
        //              polygonGeom = JGeometry.createLinearPolygon(
        //                                      pointArray, dim, dbSrid);
        //      }
        if (!pointList.isEmpty()) {
                String geomEWKT="SRID=" + dbSrid + ";POLYGON((";

        //      int dim = is2d ? 2 : 3;

                // if we have to return a 2d polygon we first have to correct the
                // double lists we retrieved from citygml4j as they are always 3d
                if (is2d) {
                        for (List<Double> coordsList : pointList) {
```

```
                        Iterator<Double> iter = coordsList.iterator();
                        int count = 0;
                        while (iter.hasNext()) {
                              iter.next();

                              if (count++ == 2) {
                                    count = 0;
                                    iter.remove();
                              }
                        }
                  }
            }

            for (List<Double> coordsList : pointList) {
                  for (int i=0; i<coordsList.size(); i+=3){
                        geomEWKT = geomEWKT + coordsList.get(i) + " " +
                          coordsList.get(i+1) + " " + coordsList.get(i+2) + ",";
                  }
                  geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
                  geomEWKT = geomEWKT + "),(";
            }

            geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 2);
            geomEWKT = geomEWKT + ")";

            polygonGeom = new PGgeometry(PGgeometry.geomFromString(geomEWKT));
      }
```

de.tub.citydb.modules.citygml.importer.database.content.**DBSurfaceData**

```
436   //   JGeometry geom = new JGeometry(coords.get(0), coords.get(1), dbSrid);
      //   STRUCT obj = SyncJGeometry.syncStore(geom, batchConn);
      //   psSurfaceData.setObject(15, obj);
      // } else
      //   psSurfaceData.setNull(15, Types.STRUCT, "MDSYS.SDO_GEOMETRY");
        Geometry geom = PGgeometry.geomFromString("SRID=" + dbSrid + ";POINT(" +
            coords.get(0) + " " + coords.get(1) + ")");
        PGgeometry pgGeom = new PGgeometry(geom);
        psSurfaceData.setObject(15, pgGeom);
      } else
        psSurfaceData.setNull(15, Types.OTHER, "ST_GEOMETRY");
```

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;
       }

384    // protected JGeometry geometry;
rep+   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);
```

5.2 From 3DCityDB back to CityGML

Simply said, the export works the other way around. In *Oracle* the `ResultSet` is casted into the `STRUCT` data type and then "loaded" into a `JGeometry` object. The *PostGIS* way works in a similar manner with `PGgeometry.getGeometry`. In *Oracle* `JGeometry` can easily be transferred to arrays and processed back again into list elements for creating the CityGML primitives. The ELEM_INFO_ARRAY helps to distinguish between geometric types. The *PostGIS* JDBC offers different sub classes of `Geometry.java`. `ComposedGeom` and `MultiLineString` were used for addressing child elements of composed geometries.

de.tub.citydb.modules.citygml.exporter.database.content.**DBAppearance**

```
418    // STRUCT struct = (STRUCT)rs.getObject("GT_REFERENCE_POINT");
rep+   // 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**

```
163   // double[] points = geom.getMBR();


169   // 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**

```
120   // 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**

```
95    //    public Point getPoint(JGeometry geom, boolean setSrsName) {
      //          Point point = null;
      //
      //          if (geom != null && geom.getType() == JGeometry.GTYPE_POINT) {
      //                int dimensions = geom.getDimensions();
      //                double[] pointCoord = geom.getPoint();
      //
      //                if (pointCoord != null && pointCoord.length >=
      //                      dimensions) {
      //                      point = new PointImpl();
      //
      //                      List<Double> value = new ArrayList<Double>();
      //                      for (int i = 0; i < dimensions; i++)
      //                            value.add(pointCoord[i]);
      //
      //                      DirectPosition pos = new DirectPositionImpl();
      //                      pos.setValue(value);
      //                      pos.setSrsDimension(dimensions);
      //                      if (setSrsName)
      //                            pos.setSrsName(gmlSrsName);
      //                      point.setPos(pos);
      //                }
//          }
//
//          return point;
//    }
public Point getPoint(Geometry geom, boolean setSrsName) {
      Point point = null;

      if (geom != null && geom.getType() == Geometry.POINT) {
            int dimensions = geom.getDimension();

            if (dimensions == 2) {
                  point = new PointImpl();

                  List<Double> value = new ArrayList<Double>();
                  value.add(geom.getPoint(0).getX());
                  value.add(geom.getPoint(0).getY());

                  DirectPosition pos = new DirectPositionImpl();
                  pos.setValue(value);
                  pos.setSrsDimension(dimensions);
                  if (setSrsName)
                        pos.setSrsName(gmlSrsName);
                  point.setPos(pos);
            }

            if (dimensions == 3) {
                  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());

                  DirectPosition pos = new DirectPositionImpl();
                  pos.setValue(value);
                  pos.setSrsDimension(dimensions);
                  if (setSrsName)
                        pos.setSrsName(gmlSrsName);
                  point.setPos(pos);
            }
      }

      return point;
}
```
148
```
//    public MultiPoint getMultiPoint(JGeometry geom, boolean setSrsName){
//          MultiPoint multiPoint = null;
//
//          if (geom != null) {
//                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]);
//
//                            DirectPosition pos = new
//                                                    DirectPositionImpl();
//                            pos.setValue(value);
//                            pos.setSrsDimension(dimensions);
//                            if (setSrsName)
//                                  pos.setSrsName(gmlSrsName);
//                            point.setPos(pos);
//
//                            PointProperty pointProperty = new
//                                  PointPropertyImpl();
//                            pointProperty.setPoint(point);
//
//                            multiPoint.addPointMember(pointProperty);
//                      }
//                }
//
//                else if (geom.getType() == JGeometry.GTYPE_POINT) {
//                      Point point = getPoint(geom, setSrsName);
//                      if (point != null) {
//                            PointProperty pointProperty = new
//                                              PointPropertyImpl();
//                            pointProperty.setPoint(point);
//                            multiPoint.addPointMember(pointProperty);
//                      }
//                }
//
//                if (!multiPoint.isSetPointMember())
//                      multiPoint = null;
//          }
//
//          return multiPoint;
//    }
```

```
public MultiPoint getMultiPoint(Geometry geom, boolean setSrsName) {
      MultiPoint multiPoint = null;

      if (geom != null) {
            multiPoint = new MultiPointImpl();
            int dimensions = geom.getDimension();

            if (geom.getType() == Geometry.MULTIPOINT) {
                  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);
                    }

                DirectPosition pos = new DirectPositionImpl();
                pos.setValue(value);
                pos.setSrsDimension(dimensions);
                if (setSrsName)
                    pos.setSrsName(gmlSrsName);
                point.setPos(pos);

                PointProperty pointProperty = new PointPropertyImpl();
                pointProperty.setPoint(point);

                multiPoint.addPointMember(pointProperty);
            }
            else if (geom.getType() == Geometry.POINT) {
                Point point = getPoint(geom, setSrsName);
                if (point != null) {
                    PointProperty pointProperty = new
                        PointPropertyImpl();
                    pointProperty.setPoint(point);
                    multiPoint.addPointMember(pointProperty);
                }
            }

            if (!multiPoint.isSetPointMember())
                multiPoint = null;
        }

        return multiPoint;
    }

245 //  public LineString getLineString(JGeometry geom, boolean setSrsName){
    //      LineString lineString = null;
    //
    //      if (geom != null && geom.getType() == JGeometry.GTYPE_CURVE) {
    //          int dimensions = geom.getDimensions();
    //          double[] ordinatesArray = geom.getOrdinatesArray();
    //
    //          List<Double> value = new ArrayList<Double>();
    //          for (int i = 0; i < ordinatesArray.length; i++)
    //              value.add(ordinatesArray[i]);
    //
    //          lineString = new LineStringImpl();
    //          DirectPositionList directPositionList = new
    //                                              DirectPositionListImpl();
    //                  directPositionList.setValue(value);
    //                  directPositionList.setSrsDimension(dimensions);
    //                  if (setSrsName)
    //                        directPositionList.setSrsName(gmlSrsName);
    //                  lineString.setPosList(directPositionList);
    //            }
    //            return lineString;
    //      }
    public LineString getLineString(Geometry geom, boolean setSrsName) {
        LineString lineString = null;

        if (geom != null && geom.getType() == Geometry.LINESTRING) {
            int dimensions = geom.getDimension();

            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);
                }

            lineString = new LineStringImpl();
            DirectPositionList directPositionList = new
                                                DirectPositionListImpl();
            directPositionList.setValue(value);
            directPositionList.setSrsDimension(dimensions);
            if (setSrsName)
                directPositionList.setSrsName(gmlSrsName);
            lineString.setPosList(directPositionList);
        }
        return lineString;
    }
```

```
291   //    public MultiCurve getMultiCurve(JGeometry geom, boolean setSrsName){
      //          MultiCurve multiCurve = null;
      //
      //          if (geom != null) {
      //            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]);
      //
      //                    LineString lineString = new LineStringImpl();
      //                    DirectPositionList directPositionList = new
      //                                          DirectPositionListImpl();
      //                    directPositionList.setValue(values);
      //                    directPositionList.setSrsDimension(dimensions);
      //                    if (setSrsName)
      //                      directPositionList.setSrsName(gmlSrsName);
      //
      //                    lineString.setPosList(directPositionList);
      //
      //                    CurveProperty curveProperty = new CurvePropertyImpl();
      //                    curveProperty.setCurve(lineString);
      //                    multiCurve.addCurveMember(curveProperty);
      //
      //                    curveElem = curveLimit;
      //                  }
      //              }
      //              else if (geom.getType() == JGeometry.GTYPE_CURVE) {
      //                LineString lineString = getLineString(geom, setSrsName);
      //                if (lineString != null) {
      //                  CurveProperty curveProperty = new CurvePropertyImpl();
      //                  curveProperty.setCurve(lineString);
      //                  multiCurve.addCurveMember(curveProperty);
      //              }
      //            }
      //
      //            if (!multiCurve.isSetCurveMember())
      //                        multiCurve = null;
      //            }
      //
      //            return multiCurve;
      //      }
      public MultiCurve getMultiCurve(Geometry geom, boolean setSrsName) {
        MultiCurve multiCurve = null;

        if (geom != null) {
          multiCurve = new MultiCurveImpl();
          int dimensions = geom.getDimension();

          if (geom.getType() == Geometry.MULTILINESTRING) {
            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);
              }

            LineString lineString = new LineStringImpl();
            DirectPositionList directPositionList = new
                                                   DirectPositionListImpl();

            directPositionList.setValue(values);
            directPositionList.setSrsDimension(dimensions);
            if (setSrsName)
              directPositionList.setSrsName(gmlSrsName);
              lineString.setPosList(directPositionList);

              CurveProperty curveProperty = new CurvePropertyImpl();
              curveProperty.setCurve(lineString);
              multiCurve.addCurveMember(curveProperty);
          }
        }
        else if (geom.getType() == Geometry.LINESTRING) {
          LineString lineString = getLineString(geom, setSrsName);
          if (lineString != null) {
            CurveProperty curveProperty = new CurvePropertyImpl();
            curveProperty.setCurve(lineString);
            multiCurve.addCurveMember(curveProperty);
          }
        }

        if (!multiCurve.isSetCurveMember())
          multiCurve = null;
        }

        return multiCurve;
    }
493 //    public Polygon getPolygon(JGeometry geom, boolean setSrsName) {
    //      Polygon polygon = null;
    //
    //      if (geom != null && geom.getType() == JGeometry.GTYPE_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) {
//              LinearRing linearRing = new LinearRingImpl();
//              DirectPositionList directPositionList = new
//                                                DirectPositionListImpl();
//
//              directPositionList.setValue(values);
//              directPositionList.setSrsDimension(dimensions);
//              if (setSrsName)
//                directPositionList.setSrsName(gmlSrsName);
//
//              linearRing.setPosList(directPositionList);
//
//              Exterior exterior = new ExteriorImpl();
//              exterior.setRing(linearRing);
//              polygon.setExterior(exterior);
//
//              isExterior = false;
//            } else {
//                LinearRing linearRing = new LinearRingImpl();
//                DirectPositionList directPositionList = new
//                  DirectPositionListImpl();
//
//                directPositionList.setValue(values);
//                directPositionList.setSrsDimension(dimensions);
//                if (setSrsName)
//                  directPositionList.setSrsName(gmlSrsName);
//                  linearRing.setPosList(directPositionList);
//                  Interior interior = new InteriorImpl();
//                  interior.setRing(linearRing);
//                  polygon.addInterior(interior);
//              }
//          }
//      }
//      return polygon;
//    }
public Polygon getPolygon(Geometry geom, boolean setSrsName) {
  Polygon polygon = null;

  if (geom != null && geom.getType() == Geometry.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) {
          LinearRing linearRing = new LinearRingImpl();
          DirectPositionList directPositionList = new
                                                DirectPositionListImpl();
          directPositionList.setValue(values);
          directPositionList.setSrsDimension(dimensions);
          if (setSrsName)
              directPositionList.setSrsName(gmlSrsName);

          linearRing.setPosList(directPositionList);
          Exterior exterior = new ExteriorImpl();
          exterior.setRing(linearRing);
          polygon.setExterior(exterior);
        } else {
        //isInterior
          LinearRing linearRing = new LinearRingImpl();
          DirectPositionList directPositionList = new
                                                DirectPositionListImpl();
          directPositionList.setValue(values);
          directPositionList.setSrsDimension(dimensions);
          if (setSrsName)
            directPositionList.setSrsName(gmlSrsName);
            linearRing.setPosList(directPositionList);
                Interior interior = new InteriorImpl();
                interior.setRing(linearRing);
                polygon.addInterior(interior);
              }
          }
      }
    return polygon;
  }
```

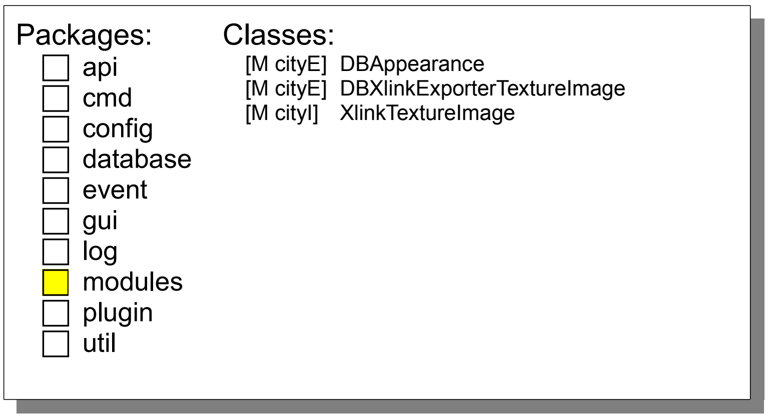
de.tub.citydb.util.database.**DBUtil**

```
308   // STRUCT struct = (STRUCT)rs.getObject(1);
rep+  // 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();
```

5.3 Synchronization of geometric functions

It is proven that the method `store(JGeometry)` is not threadsafe and deadlocks can occur. This problem is avoided by synchronizing the storing of `JGeometries` into `STRUCT` objects with a Java `Reentrant Lock` (inside `SyncJGeometry.java`). Until now no such problem occurred for the *PostGIS* version.

6. How to deal with textures and binary data



As the ORDImage data type differs a lot from the BYTEA data type in *PostgreSQL* it is not surprising that the im- and export of textures had to be changed in many aspects. With ORDImage it is possible to query metadata from the images and also use functions similar to a graphic processing software. Some of these features are called in the `DBAppearance` class (see also chapter **3.3**). Overall, the *3DCityDB* hardly uses the abilities of ORDImage. Even Oracle itself recommended the use of BLOBs for the *3DCityDB* to the developers.

de.tub.citydb.modules.citygml.exporter.database.content.**DBAppearance**
```
138    // nvl(sd.TEX_IMAGE.getContentLength(), 0) as DB_TEX_IMAGE_SIZE,
rep    // 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,
```

The function `ORDImage.getMimeType` could not be translated. The information from column `sd.TEX_MIME_TYPE` has to do it.

The exchange of binary data also applies for Library Objects in the table `Implicit_Geometry`. The implementation in Java is identical. Some differences invoke from the use of the BLOB data type in the *Oracle* version (see 6.1).

Next to the handling of BYTEA objects in Java an alternative routine was programmed for *PostgreSQL* Large Objects (LOB). The main disadvantage of the BYTEA data type is that it can not be parsed sequentially into a Java stream object. The whole object has to be loaded into the RAM. There is no risk when dealing with single texture files as they are usually very small in size but when using texture atlases the Java Heap Space might be exceeded quite fast. A texture atlas is a collection of textures in one file. Parts of the image can be addressed with texture coordinates. This is very useful if a certain texture is referenced to multiple objects like for example windows. But every time a window texture is queried the whole texture atlas would have to be read into the main memory.

LOBs would not have such a problem as they can be read sequentially. In all our test cases it could not be proven that the usage of LOBs accelerates the im- and exports. Another difference to the BYTEA data type appears for the storage of data. LOBs are stored separately in the internal file system of *PostgreSQL*. The corresponding column in the table only holds an object identifier (OID) that points to the actual objects. When using BYTEA the whole binary file is stored in the column. That is why it could take some time if a full table scan is executed on big table containing BYTEA objects.

The deletion of those OIDs values does not remove their referenced LOBs. Therefore functions like `vacuumlo` or `lo_unlink` have to be used. To save the user the more complicated handling of LOBs the storage of BYTEA is still preferred for the release version.

6.1 Import of textures

As seen on the following examples the code for importing textures could be reduced to a few lines. Inserting ORDImages works as follows (also see figure 3 on page 40):

1. initialization of the target column with `ordimage.init()`
2. a SELECT FOR UPDATE query locks the `ResultSet` cursor for the row to be updated
3. the database ORDImage is transferred into a ORDImage Java object but still empty
4. `loadDataFromInputStream` fills the empty ORDImage Java object
5. `setORAData` sets the ORDImage Java object in the `PreparedStatement` which inserts the data by updating the table `Surface_Data`

With BLOBs the output of the `InputStream` can directly be set in the `PreparedStatement` with `setBinaryStream`.

de.tub.citydb.modules.citygml.importer.database.xlink.resolver.**XlinkTextureImage**

```
// 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=?");
```

```
// // 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;
      //    }

      // 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;
```

6.2 Export of textures

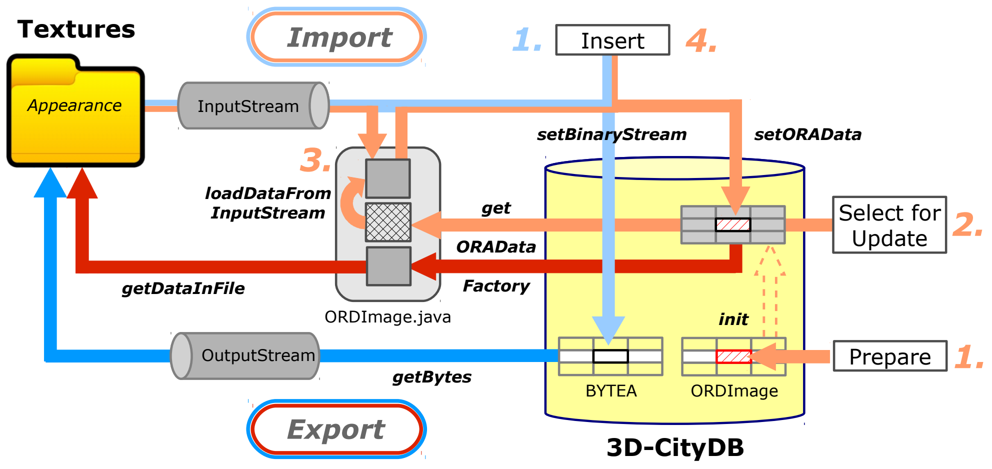


Fig. 3: Im- und Export of textures (Kunde 2012 [2])

de.tub.citydb.modules.citygml.exporter.database.xlink.**DBXlinkExporterTextureImage**
de.tub.citydb.modules.citygml.exporter.database.xlink.**DBXlinkExporterLibraryObject**

```
128   // OracleResultSet rs = (OracleResultSet)psTextureImage.executeQuery();
rep+  ResultSet rs = (ResultSet)psTextureImage.executeQuery();

143   // // read oracle image data type
rep+  // 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();
      // }
```

Used method:

```
      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;
        }
```

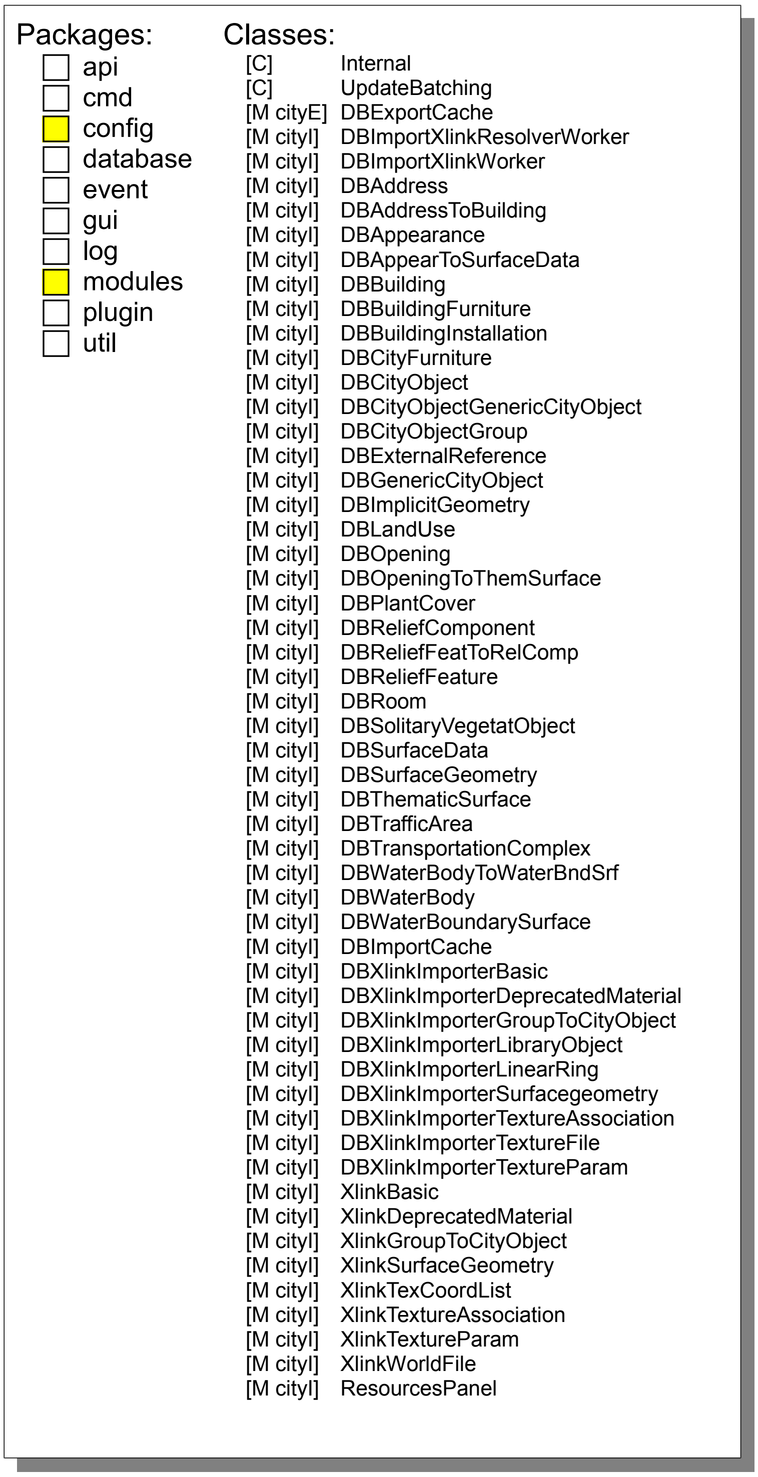
Alternative way:

```
        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 method:

```
        // 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();
```

7. The batchsize of PostgreSQL



Packages:

- ☐ api
- ☐ cmd
- 🟨 config
- ☐ database
- ☐ event
- ☐ gui
- ☐ log
- 🟨 modules
- ☐ plugin
- ☐ util

Classes:

[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

The maximum batchsize of *PostgreSQL* was set to 10000 and given the name *POSTGRESQL_MAX_BATCH_SIZE*. A higher value might be possible but was not tested. The parameter had to be renamed in many classes.

de.tub.citydb.config.internal.**Internal**

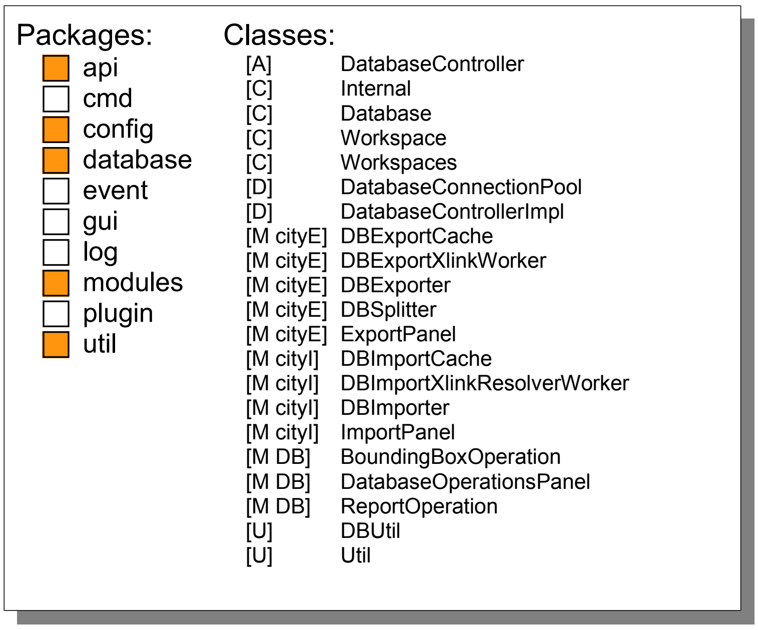
```
39    //    public static final int ORACLE_MAX_BATCH_SIZE = 65535;
      public static final int POSTGRESQL_MAX_BATCH_SIZE = 10000;
```

In the following examples no equivalent methods could be found for the Java `PreparedStatement`. The `psDrain` batch is now executed and not sent.

de.tub.citydb.modules.citygml.exporter.database.gmlid.**DBExportCache**
de.tub.citydb.modules.citygml.importer.database.gmlid.**DBImportCache**

```
82    // ((OraclePreparedStatement)psDrains[i]).setExecuteBatch(batchSize);

143   // ((OraclePreparedStatement)psDrain).sendBatch();
      psDrain.executeBatch();
```

8. Workspace Management



Packages:

- 🟧 api
- ☐ cmd
- 🟧 config
- 🟧 database
- ☐ event
- ☐ gui
- ☐ log
- 🟧 modules
- ☐ plugin
- 🟧 util

Classes:

[A]	DatabaseController
[C]	Internal
[C]	Database
[C]	Workspace
[C]	Workspaces
[D]	DatabaseConnectionPool
[D]	DatabaseControllerImpl
[M cityE]	DBExportCache
[M cityE]	DBExportXlinkWorker
[M cityE]	DBExporter
[M cityE]	DBSplitter
[M cityE]	ExportPanel
[M cityI]	DBImportCache
[M cityI]	DBImportXlinkResolverWorker
[M cityI]	DBImporter
[M cityI]	ImportPanel
[M DB]	BoundingBoxOperation
[M DB]	DatabaseOperationsPanel
[M DB]	ReportOperation
[U]	DBUtil
[U]	Util

PostgreSQL does not offer a workspace or history management like *Oracle* does. Every part in the Java code concerning these workspace features was uncommented but not deleted as there might be a solution for database versioning in the future. The affected packages are colored orange.

9. KML-Exporter

Packages:	Classes:	
☐ api	[M kml]	KmlExportWorker
☐ cmd	[M kml]	KmlExporter
☐ config	[M kml]	BalloonTemplateHandlerImpl
☐ database	[M kml]	CityObjectGroup
☐ event	[M kml]	ColladaBundle
☐ gui	[M kml]	KmlExporterManager
☐ log	[M kml]	KmlGenericObject
🟨 modules	[M kml]	KmlSplitter
☐ plugin	[M kml]	Queries
☐ util		

Due to the modular architecture of the *Importer/Exporter* the port of the *KML-Exporter* only affected classes of the KML module. The code design differs from the CityGML module.

9.1 Queries

Database queries are collected in one central class and were used as string constants in other classes.

de.tub.citydb.modules.kml.database.**Queries**

```
44    // public static final String GET_IDS =
rep   //    "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_IDS =
      "SELECT co.id, co.gmlid, co.class_id " +
      "FROM CITYOBJECT co " +
      "WHERE " +
            "ST_Intersects(co.envelope, ST_GeomFromEWKT(?)) = 'TRUE' " +
      "UNION ALL " +
            "SELECT co.id, co.gmlid, co.class_id " +
            "FROM CITYOBJECT co " +
            "WHERE " +
              "ST_CoveredBy(co.envelope, ST_GeomFromEWKT(?)) = 'TRUE' " +
      "ORDER BY 3"; // ORDER BY co.class_id*/

56    //    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";
```

The following query is only needed in the *PostGIS* version to prevent a full table scan of the `Surface_Data` table as it holds BYTEA objects (see chapter 6).

```
76    public static final String GET_TEXIMAGE_FROM_SURFACE_DATA_ID =
            "SELECT sd.tex_image " +
            "FROM SURFACE_DATA sd " +
            "WHERE " +
                  "sd.id = ?";

86    //    public static final String TRANSFORM_GEOMETRY_TO_WGS84 =
rep   //          "SELECT SDO_CS.TRANSFORM(?, 4326) FROM DUAL";
      public static final String TRANSFORM_GEOMETRY_TO_WGS84 =
            "SELECT ST_Transform(?, 4326)";

112   //    public static final String GET_CENTROID_LAT_IN_WGS84_FROM_ID =
rep   //      "SELECT v.Y FROM TABLE(" +
      //         "SELECT SDO_UTIL.GETVERTICES(
      //            SDO_CS.TRANSFORM(
      //              SDO_GEOM.SDO_CENTROID(co.envelope, 0.001), 4326)) " +
      //                "FROM CITYOBJECT co " +
      //                   "WHERE co.id = ?) v";
      public static final String GET_CENTROID_LAT_IN_WGS84_FROM_ID =
            "SELECT ST_Y(ST_Transform(ST_Centroid(co.envelope), 4326)) " +
            "FROM CITYOBJECT co " +
            "WHERE co.id = ?";

172   //    public static final String GET_ENVELOPE_HEIGHT_MIN_IN_WGS84_FROM_ID
rep   //    =
      //    "SELECT SDO_GEOM.SDO_MIN_MBR_ORDINATE(
      //    SDO_CS.TRANSFORM(co.envelope, 4326), 3) " +
      //          "FROM CITYOBJECT co " +
      //          "WHERE co.id = ?";
      public static final String GET_ENVELOPE_HEIGHT_MIN_IN_WGS84_FROM_ID =
            "SELECT ST_ZMin(Box3D(ST_Transform(co.envelope, 4326))) " +
            "FROM CITYOBJECT co " +
            "WHERE co.id = ?";
```

The following example is a bit tricky. In *Oracle* a best practice to aggregate a large number of polygons is to perform a sort of pyramid aggregation. That means aggregations are primarily done on smaller adjacent groups which are then aggregated to bigger groups and so on (see GROUP BY clauses at the end of the query). Internally the *PostGIS* function ST_Union works similar using the CascadedPolygonUnion function of the *GEOS* library. Thus GROUP BY aggregations are not needed.

```
454   //    public static final String
rep   //    BUILDING_PART_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>) " +
     //          ")";
     public static final String
     BUILDING_PART_GET_AGGREGATE_GEOMETRIES_FOR_LOD2_OR_HIGHER =
           "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 BUILDING b " +
                 "WHERE b.building_root_id = ? " +
                 "AND b.lod<LoD>_geometry_id IS NOT NULL " +
           "UNION " +
           "SELECT ts.lod<LoD>_multi_surface_id " +
           "FROM BUILDING b, THEMATIC_SURFACE ts " +
           "WHERE b.building_root_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 " +
           // ST_Area for WGS84 only works correctly if the geometry is a
           // geography data type
           "WHERE ST_Area(ST_Transform(get_valid_geoms.simple_geom,4326)
                 ::geography, true) > <TOLERANCE>) AS get_valid_area";

651  //    private static final String SOLITARY_VEGETATION_OBJECT_COLLADA_ROOT_
rep  //    IDS= "SELECT ? FROM DUAL "; // dummy
     private static final String SOLITARY_VEGETATION_OBJECT_COLLADA_ROOT_IDS =
           "SELECT ?"; // dummy
```

de.tub.citydb.modules.kml.database.**KmlSplitter**

```
187  //    BoundingBox tile =
rep  //          exportFilter.getBoundingBoxFilter().getFilterState();
     //    OracleResultSet rs = null;
     //    PreparedStatement spatialQuery = null;
     //    try {
     //          spatialQuery =
     //          connection.prepareStatement(TileQueries.QUERY_GET_IDS);
     //          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();
     BoundingBox tile = exportFilter.getBoundingBoxFilter().getFilterState();
     ResultSet rs = null;
     PreparedStatement spatialQuery = null;
     String lineGeom = null;
     String polyGeom = null;

     try {
           spatialQuery = connection.prepareStatement(Queries.GET_IDS);
           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() + "))";

           spatialQuery.setString(1, lineGeom);
           spatialQuery.setString(2, polyGeom);

           rs = spatialQuery.executeQuery();
```

The `BallonTemplateHandlerImpl` class builds up a queries for the KML balloon content. Most of them are aggregated queries. If multiple rows are fetched by the `ResultSet` and no aggregation was used one row has to be picked. Therefore the window function ROW_NUMBER() was used. As *PostgreSQL* does not allow the usage of window function inside of a WHERE clause the queries have to be re-written in a more nested way (except for the first example, that did not need a range condition for rnum like in *Oracle*).

de.tub.citydb.modules.kml.database.**BalloonTemplateHandlerImpl**

```
// if (rownum > 0) {
//    sqlStatement = "SELECT * FROM (SELECT a.*, ROWNUM rnum FROM (" +
//                      sqlStatement + " ORDER by " + tableShortId + "." +
//                      columns.get(0) + " ASC) a WHERE ROWNUM <= " +
//                      rownum + ") 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";
// }
if (rownum > 0) {
  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 " +
                    "(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";
}
```

9.2 Geometries for KML placemarks

Most of the changes were similar to examples in chapter 5 and more or less self-explaining. The `JGeometry.getOrdinatesArray()` method is substituted with a simple iteration to fill the array. Some extra variables and *PostGIS* JDBC classes (and their methods) are used to port *Oracle*'s `ELEM_INFO_ARRAY` methods properly. `KMLGenericObject.java` is the super class to the export classes covering the thematic modules of CityGML.

de.tub.citydb.modules.kml.database.**KmlGenericObject**

```
1469   // PolygonType polygon = null;
rep+   // while (rs.next()) {
       //   STRUCT buildingGeometryObj = (STRUCT)rs.getObject(1);
       //   if (!rs.wasNull() && buildingGeometryObj != null) {
       //     eventDispatcher.triggerEvent(new GeometryCounterEvent(null, this));
       //
       //      polygon = kmlFactory.createPolygonType();
       //      polygon.setTessellate(true);
       //      polygon.setExtrude(false);
       //      polygon.setAltitudeModeGroup(
       //                      kmlFactory.createAltitudeMode(
       //                            AltitudeModeEnumType.CLAMP_TO_GROUND));
       //
       //      JGeometry groundSurface = 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:
       //                                  polygon.setOuterBoundaryIs(boundary);
       //                                  break;
       //                            case INTERIOR_POLYGON_RING:
       //                                  polygon.getInnerBoundaryIs().
       //                                                          add(boundary);
       //                                  break;
       //                            case POINT:
       //                            case LINE_STRING:
       //                                  continue;
       //                            default:
       //                                  Logger.getInstance().warn("Unknown
       //                                    geometry for " + work.getGmlId());
       //                                  continue;
       //      }
       PolygonType polygon = null;
       PolygonType[] multiPolygon = null;
       while (rs.next()) {
         PGgeometry pgBuildingGeometry = (PGgeometry)rs.getObject(1);

         if (!rs.wasNull() && pgBuildingGeometry != null) {
           eventDispatcher.triggerEvent(new GeometryCounterEvent(null, this));
```

```
      polygon = kmlFactory.createPolygonType();
      polygon.setTessellate(true);
      polygon.setExtrude(false);
      polygon.setAltitudeModeGroup(kmlFactory.createAltitudeMode(
                                   AltitudeModeEnumType.CLAMP_TO_GROUND));

    Geometry groundSurface=convertToWGS84(pgBuildingGeometry.getGeometry());
    switch (groundSurface.getType()) {
    case Geometry.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;
              }
              ...
    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),etc.)
            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]));
            }
        }

1794  // for (int i = 0; i < surface.getElemInfo().length; i = i+3) {
rep+  //    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 = probablyRoof &&
      //      (reducePrecisionForZ(ordinatesArray[j+2] - lowestZCoordinate)>0);
      //      // not touching the ground
      //
      //      if (currentLod == 1) { // calculate normal
      //        int current = j;
      //        int next = j+3;
      //        if (next >= startNextRing) next = surface.getElemInfo()[i] - 1;
      //          nx = nx + ((ordinatesArray[current+1]-ordinatesArray[next+1])
      //          * (ordinatesArray[current+2] + ordinatesArray[next+2]));
      //          ny = ny + ((ordinatesArray[current+2]-ordinatesArray[next+2])
      //          * (ordinatesArray[current] + ordinatesArray[next]));
      //          nz = nz + ((ordinatesArray[current] - ordinatesArray[next])
      //          * (ordinatesArray[current+1] + ordinatesArray[next+1]));
      //      }
      // }}
      int cellCount = 0;
      for (int i = 0; i < surface.numRings(); i++){
        LinearRingType linearRing = kmlFactory.createLinearRingType();
        BoundaryType boundary = kmlFactory.createBoundaryType();
        boundary.setLinearRing(linearRing);
        if (i == 0) { // EXTERIOR_POLYGON_RING
          polygon.setOuterBoundaryIs(boundary);
        }
        else { // INTERIOR_POLYGON_RING
          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 = probablyRoof && (reducePrecisionForZ(ordinatesArray[j+2]
                       - lowestZCoordinate) > 0);
      // not touching the ground

      if (currentLod == 1) { // calculate normal
        int current = j;
        int next = j+3;
        if (next >= ordinatesArray.length) next = 0;
            nx = nx + ((ordinatesArray[current+1] - ordinatesArray[next+1]) *
                  (ordinatesArray[current+2] + ordinatesArray[next+2]));
            ny = ny + ((ordinatesArray[current+2] - ordinatesArray[next+2]) *
                  (ordinatesArray[current] + ordinatesArray[next]));
            nz = nz + ((ordinatesArray[current] - ordinatesArray[next]) *
                  (ordinatesArray[current+1] + ordinatesArray[next+1]));
        }
      }
      cellCount += (surface.getRing(i).numPoints()*3);
```

9.3 Textures for COLLADA export

The database can store texture formats that are unknown to ORDImage. Therefore two methodologies were implemented in the *KML-Exporter*. One to deal with ORDImages and another to process all the unknown formats as BLOBs. Fortunately the last one could be used for the *PostGIS* port. All the TexOrdImage methods had to be uncommented from the following classes and the texture export for COLLADA exports was slightly changed.

de.tub.citydb.modules.kml.database.**KmlGenericObject**

```
1930  //    OrdImage texImage = null;
      InputStream texImage = null;

1954  addTexImageUri(surfaceId, texImageUri);
      // if (getTexOrdImage(texImageUri) == null) { // not already marked as
                                                  wrapping texture
```

Additional query to get textures:

```
1960  psQuery3 = connection.prepareStatement(Queries.
                                GET_TEXIMAGE_FROM_SURFACE_DATA_ID);
      psQuery3.setLong(1, rs2.getLong("surface_data_id"));
      rs3 = psQuery3.executeQuery();
      while (rs3.next()) {
        /*
        // read large object (OID) data type from database
        // Get the Large Object Manager to perform operations with
        LargeObjectManager lobj = ((org.postgresql.PGConnection)connection).
                                  getLargeObjectAPI();

        // Open the large object for reading
        long oid = rs3.getLong("tex_image");
        if (oid == 0) {
          Logger.getInstance().error(
            "Database error while reading library object: " + texImageUri);
        }
        LargeObject obj = lobj.open(oid, LargeObjectManager.READ);

        // Read the data
        buf = new byte[obj.size()];
        obj.read(buf, 0, obj.size());
        */
        // read bytea data type from database
        texImage = rs3.getBinaryStream("tex_image");
      }

1996  //bufferedImage = ImageIO.read(texImage.getDataInStream());
      bufferedImage = ImageIO.read(texImage);

2003  // else {
      //        addTexOrdImage(texImageUri, texImage);
      //       }
      // }

2064  /* if (s > 1.1 || s < -0.1 || t < -0.1 || t > 1.1) {
      // 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);
        }
      */
```

de.tub.citydb.modules.kml.concurrent.**KmlExportWorker**
de.tub.citydb.modules.kml.controller.**KmlExporter**
de.tub.citydb.modules.kml.database.**CityFurniture**
de.tub.citydb.modules.kml.database.**ColladaBundle**
de.tub.citydb.modules.kml.database.**GenericCityObject**
de.tub.citydb.modules.kml.database.**SolitaryVegetationObject**
de.tub.citydb.modules.kml.database.**KmlExporterManager**

```
rep+  //    uncommented TexOrdImage-methods
```

10. References

Documents:

[1] KUNDE, F. ; ASCHE, H. ; KOLBE, T.H. ; NAGEL, C. ; HERRERUELA, J. ; KÖNIG, G. (2013): 3D City Database for CityGML: Port documentation: PL/SQL to PL/pgSQL.
Accessible under:
http://opportunity.bv.tu-berlin.de/software/projects/3dcitydb-imp-exp/documents

[2] KUNDE, F. (2012): CityGML in PostGIS – Portierung, Anwendung und Performanz-Analyse am Beispiel der 3D City Database von Berlin. Master Thesis (in german only).
Accessible under: Link following soon at www.3dcitydb.net.

Links:

www1	http://docs.oracle.com/cd/E14072_01/java.112/e12826/toc.htm
www2	http://commons.apache.org/dbcp/api-1.4/index.html
www3	http://tomcat.apache.org/tomcat-7.0-doc/api/index.html
www4	http://www.mchange.com/projects/c3p0/apidocs/index.html
www5	http://spatialreference.org

List of figures: