Portierung der 3D-City-DB von Oracle Spatial nach PostGIS

Anpassungen am Importer-Exporter-Tool

(von Felix Kunde)

---

0. Eine kleine Legende

Die Box-Übersichten beziehen sich jeweils auf den Inhalt des Kapitels. Sie sollen einen schnellen Überblick geben, welche Klassen angepasst werden mussten und auf welche Projektpakete sich diese verteilt haben.

Pakete:

☐ api	= Inhalte mussten nicht angepasst werden
☐ database	= einige Inhalte wurden angepasst
☐ modules	= einige Inhalte müssten zukünftig angepasst werden (z.B. Connection Pooling)

Klassen:

[A]	aus dem Paket 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		

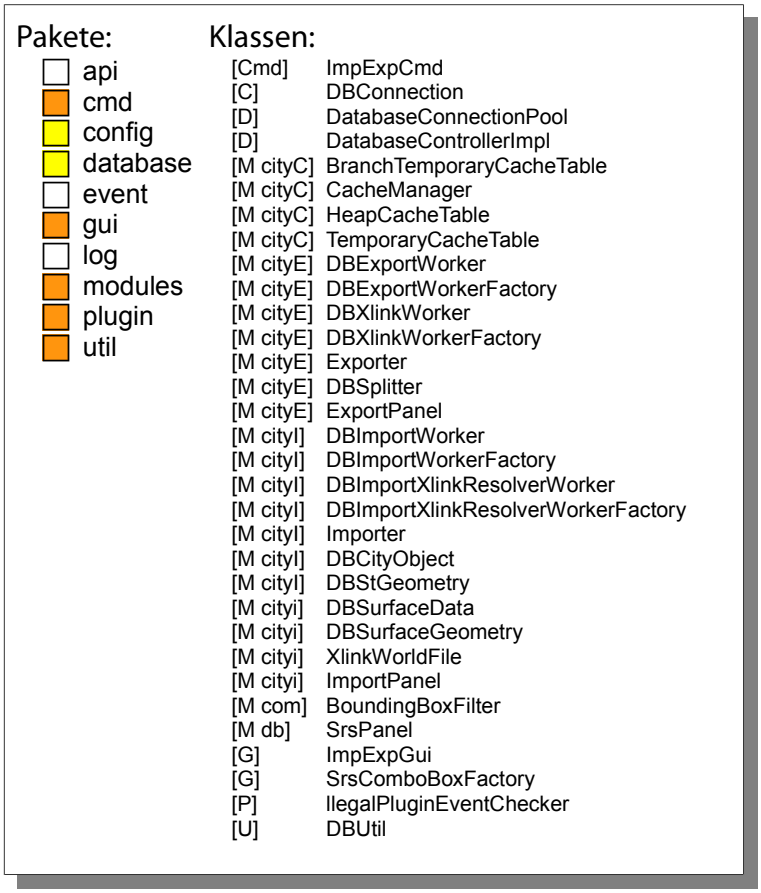
Quellcodezeilen:

`59`	Änderung findet sich ab Zeile 59 in entsprechender Java-Klasse
`115+`	Diese und folgende Zeilen sind auskommentiert. Sie wurden nicht umgesetzt, weil es nicht möglich oder nicht notwendig war
`wdh`	Dieses Code-Beispiel wiederholt sich in der selben Klasse.
`wdh+`	Dieses Code-Beispiel wiederholt sich in der selben Klasse und in anderen Klassen.

```
//private Integer port = 1521;
```
auskommentierter Oracle-spezifischer Code (bereits aus der Klasse entfernt)

```
private Integer port = 5432;
```
PostGIS-spezifischer Code

1. Die Verbindung zur Datenbank



Um eine Verbindung zu einer PostgreSQL-Datenbank herzustellen bedarf es nicht vieler Änderungen, vorrausgesetzt man nutzt weiterhin den Universal Connection Pool von Oracle. Die PoolDataSource des UCP muss auf die DataSource von PostgreSQL (PGSimpleDataSource) zugreifen können und die URL muss an den JDBC-Treiber von PostgreSQL addressiert sein. Es war erforderlich den Datenbank-Namen separat zu definieren. Die Übergabe mittels conn.getSid() innerhalb der URL reichte nicht. Der Absatz rund um die Connection-Properties wurde ausgeklammert, da die PostgreSQL-eigene Connection-Klasse nur über die gleichen Attribute wie die Java-Connection-Klasse verfügte (d.h. `CONNECTION_PROPERTY_USE_THREADLOCAL_BUFFER_CACHE` war nicht verfügbar).

Leider entspricht dieses Vorgehen nicht dem Anspruch einer vollständigen Umsetzung auf OpenSource, weshalb zwei freie Alternativen (Apache Jakarta DBCP und C3PO) sowie eine einfach Verbindung über den DriverManager auf ihre Anwendbarkeit untersucht und getestet wurden. Da sich aber kein schneller Erfolg einstellen wollte, wurde weiterhin dem UCP von Oracle vertraut. Die Klassen-Übersicht der Box verdeutlich außerdem wie die Methoden der DatabaseConnectionPool-Klasse versteut sind. Oft handelt es sich zwar nur um die Methode getConnection(), dennoch waren Anpassungsversuche zeitaufwändig . Die betroffenen Klassen sind hier nicht aufgeführt, weil letztendlich keine Veränderungen stattfanden.

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

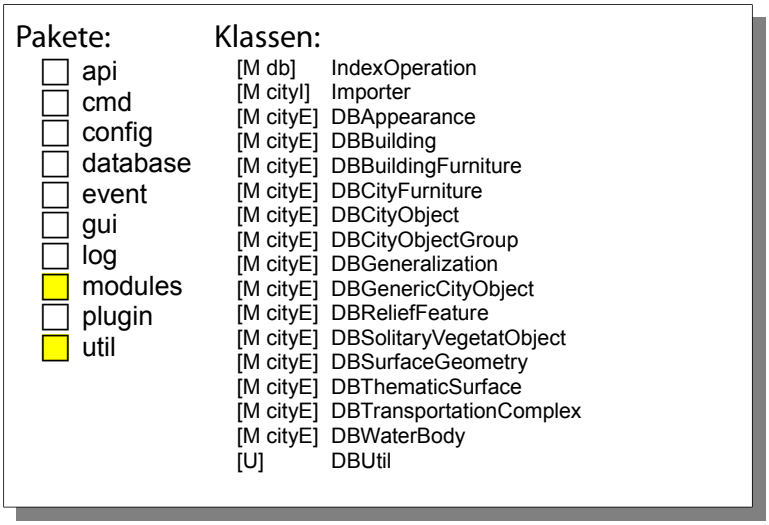
```
59    //private Integer port = 1521;
      private Integer port = 5432;

180   // für KML-Export (muss evtl. gar nicht angepasst werden)
      // return user + "@" + server + ":" + port + "/" + sid;
```

```
        return user + "://" + server + ":" + port + "/" + sid;
```

de.tub.citydb.database.**DatabaseConnectionPool**

```
59      //private final String poolName = "oracle.pool";
        private final String poolName = "postgresql.pool";

109     // poolDataSource.setConnectionFactoryClassName(
        //     "oracle.jdbc.pool.OracleDataSource");
        poolDataSource.setConnectionFactoryClassName(
              "org.postgresql.ds.PGSimpleDataSource");

110     poolDataSource.setDatabaseName(conn.getSid());

111     // poolDataSource.setURL("jdbc:oracle:thin:@//" + conn.getServer() + ":" +
              conn.getPort()+ "/" + conn.getSid());
        poolDataSource.setURL("jdbc:postgresql://" + conn.getServer() + ":" +
              conn.getPort() + "/" + conn.getSid());

115+    // set connection properties
```

2. Das Abrufen der PL/pgSQL-Funktionen über Buttons



Hinter den Funktionalitäten im Datenbank-Panel des Importer-Exporters stecken größtenteils selbstgeschriebene Prozeduren, die in der Datenbank gespeichert sind. Die wichtigen Anpassungen für PostgreSQL sind also schon mit dem Übersetzen der PL-Skripte geschehen. In Java mussten nur noch die Funktionsnamen ausgetauscht werden.

2a. Index-Funktionen, Datenbank-Report, Util-Funktionen innerhalb von Statements

Bei den Index-Funktionen wurde ein alternativer Weg zur bisherigen Verfahrensweise entwickelt. Falls der Index wie bisher gedropt wird, bleiben anders als in Oracle keine Index-Metadaten zurück die wiederverwendet werden können. In PostgreSQL ist es allerdings möglich, einen Index auf INVALID zu setzten, so dass er bei eingehenden Importen inaktiv bleibt, auch wenn er noch vorhanden ist. Ob dies die Performance steigert, muss noch getestet werden. Sollte die „Switch“-Methode verwendet werden müsste neben den anderen Funktionsnamen auch die Klassen IndexOperation und Importer angepasst werden.

de.tub.citydb.modules.database.gui.operations.**IndexOperation**
de.tub.citydb.modules.citygml.importer.controller.**Importer**

Drop-Variante

```
301   // if (!parts[4].equals("DROPPED")) {
wdh+  Switch-Variante
      if (!parts[4].equals("INVALID")) {
```

für alle de.tub.citydb.modules.citygml.exporter.database.content.**DB***

```
      //geodb_util.transform_or_null(...
      geodb_pkg.util_transform_or_null(...
```

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

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

91    // rs = stmt.executeQuery("select * from table(geodb_util.db_metadata)");
      rs = stmt.executeQuery("select * from geodb_pkg.util_db_metadata() as t");

199   // callableStmt = (OracleCallableStatement)conn.prepareCall("{? = call
wdh   //    geodb_stat.table_contents}");
      callableStmt = (CallableStatement)conn.prepareCall("{? = call
            geodb_pkg.stat_table_contents()}");

200   // callableStmt.registerOutParameter(1, OracleTypes.ARRAY, "STRARRAY");
wdh   callableStmt.registerOutParameter(1, Types.ARRAY);

203   // ARRAY result = callableStmt.getARRAY(1);
wdh   Array result = callableStmt.getArray(1);

~400  // String call = type == DBIndexType.SPATIAL ?
wdh   //          "{? = call geodb_idx.drop_spatial_indexes}" :
      //                "{? = call geodb_idx.drop_normal_indexes}";
      String call = type == DBIndexType.SPATIAL ?
            "{? = call geodb_pkg.idx_drop_spatial_indexes()}" :
                  "{? = call geodb_pkg.idx_drop_normal_indexes()}";
```

bzw. Switch-Funktion:

```
      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);

~590  // callableStmt = (OracleCallableStatement)conn.prepareCall("{? = call
      //    geodb_util.error_msg(?)}");
      callableStmt = (CallableStatement)conn.prepareCall("{? = call
            geodb_pkg.util_error_msg(?)}");
```

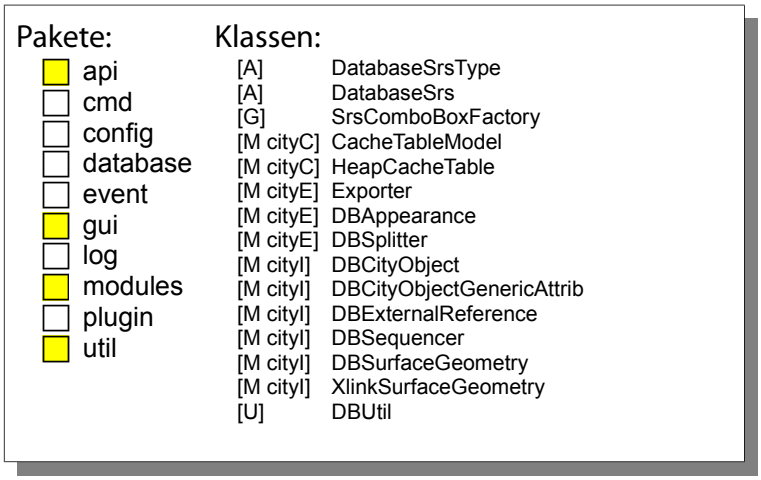
2b. Die Berechnung der BoundingBox

Bei der BoundingBox-Berechnung wurde zunächst die Workspace-Variable ausgeklammert. Der Abfrage-String musste equivalente PostGIS-Funktionen aufrufen (sdo_aggr_mbr → ST_Extent, geodb_util.to2d → ST_Force_2d). Bei der Transformation müssen 5 Punkte angegeben werden, weil PostGIS nicht die Kurzschreibweise für Rechtecke mit ledigleich zwei Punkten (LLB, URT) unterstützt.

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

```
237   // public static BoundingBox calcBoundingBox(Workspace workspace,
      //    FeatureClassMode featureClass) throws SQLException {
      public static BoundingBox calcBoundingBox(FeatureClassMode featureClass)
            throws SQLException {

249   // 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";

314   // 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);

641   // 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 = conn.prepareStatement("select ST_TRANSFORM(
      select ST_GeomFromText('POLYGON((? ?,? ?,? ?,? ?,? ?))'," +
      sourceSrid + ")," + targetSrid + ")");

643   // 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.setDouble(1, bbox.getLowerLeftCorner().getX());
      psQuery.setDouble(2, bbox.getLowerLeftCorner().getY());
      psQuery.setDouble(3, bbox.getUpperRightCorner().getX());
      psQuery.setDouble(4, bbox.getLowerLeftCorner().getY());
      psQuery.setDouble(5, bbox.getUpperRightCorner().getX());
      psQuery.setDouble(6, bbox.getUpperRightCorner().getY());
      psQuery.setDouble(7, bbox.getLowerLeftCorner().getX());
      psQuery.setDouble(8, bbox.getUpperRightCorner().getY());
      psQuery.setDouble(9, bbox.getLowerLeftCorner().getX());
      psQuery.setDouble(10, bbox.getLowerLeftCorner().getY());
```

3. Statement-Strings und Datenbank-SRS



3a. Datenbank-SRS

PostGIS unterstützt bisher keine 3D-Refrenzsysteme. Es ist unklar, ob eine mögliche zukünftige Unterstützung ähnlich stark reglementiert sein wird wie bei Oracle Spatial, d.h. ob ähnliche Abfragen wie bisher überhaupt notwendig sind. Momentan können die betroffenen Teile des Quellcodes nur ausgeklammert werden. Weiterhin sind die Unterschiede zwischen SDO_COORD_REF_SYS (Oracle) und SPATIAL_REF_SYS (PostGIS) zu beachten.

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

```
4       // PROJECTED("Projected"),
        // GEOGRAPHIC2D("Geographic2D"),
        // GEOCENTRIC("Geocentric"),
        // VERTICAL("Vertical"),
        // ENGINEERING("Engineering"),
        // COMPOUND("Compound"),
        // GEOGENTRIC("Geogentric"),
        // GEOGRAPHIC3D("Geographic3D"),
        GEOGCS("Geographic"),
        PROJCS("Projected"),
```

de.tub.citydb.api.config.**DatabaseSrs**

```
107     // public boolean is3D() {
        //     return type == DatabaseSrsType.COMPOUND || type ==
        //     DatabaseSrsType.GEOGRAPHIC3D;
        // }
```

de.tub.citydb.gui.factory.**SrsComboBoxFactory**

```
188     // if (showOnlySameDimension && refSys.is3D() != dbRefSys.is3D())
        //     continue;
```

de.tub.citydb.modules.citygml.exporter.controller.**Exporter**

```
231+  // if (internalConfig.isTransformCoordinates()) {
```

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 = ? ");

706   // if (!srs.is3D())

709+  // 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.GEOGCS ?
      "select min(crs2d.srid) from spatial_ref_sys crs3d,
            spatial_ref_sys crs2d where crs3d.srid = " + srs.getSrid() +
            " and crs2d.srtext LIKE '%GEOGCS%'" :
                  ""); --kann nicht übersetzt werden
```

3b. BoundingBox-Filter und Optimizer Hints in DBSplitter

In der Klasse DBSplitter wird bei der Oracle-Version die Funktion SDO_RELATE aufgerufen, welche für die String-Variable bboxFilter benötigt wird. Dieser BoundingBox-Filter überprüft topologische Beziehungen angelehnt an die 9-Intersektions-Matrix (DE-9IM) nach Egenhofer / Clementini. In Oracle Spatial lässt sich das Maskenattribut in SDO_RELATE mittels logischem Oder (+) kombinieren. Die Form bleibt dabei ein String. Das geht in PostGIS nicht. Die equivalente Funktion ST_RELATE erlaubt nur eine Maske in der Schreibweise der DE-9IM. Die Variable bboxFilter wurde deshalb als Liste definiert, wobei jedes Listenelement eine ST_RELATE-Abfrage mit einer anderen Intersektionsmaske beinhaltet. Für spätere Aufrufe beim Zusammensetzen des kompletten Abfrage-Strings muss daher immer komplett durch bboxFilter iteriert und die Durchläufe mit einem „ or „ verbunden werden (letztes OR durch substring eliminieren).

Eine weitere Besonderheit, die nur die Klasse DB_Splitter betrifft, ist die Verwendung von Optimizer Hints. Sie werden eingesetzt, um dem Abfrage-Optimizer Entscheidungen bei der Wahl eines optimalen Ausführungsplans abzunehmen. Da Optimizer Hints in PostgreSQL jedoch nicht eingesetzt werden, musste jedes Auftreten im Quellcode ausgeklammert werden.

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

```
82    // private String bboxFilter;
      private List<String> bboxFilter;

163   // String mask = ((tiledBBox.getTiling().getMode() != TilingMode.NO_TILING
      //    || tiledBBox.isSetOverlapMode())) ?
      //          "INSIDE+CONTAINS+EQUAL+COVERS+COVEREDBY+OVERLAPBDYINTERSECT" :
      //                "INSIDE+COVEREDBY+EQUAL";

      // bboxFilter = "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 + ")), " +
//                "'querytype=WINDOW mask=" + mask + "') = 'TRUE'";
List<String> maskType = new ArrayList<String>();

if ((tiledBBox.getTiling().getMode() != TilingMode.NO_TILING |
      tiledBBox.isSetOverlapMode())){

      maskType.add("T*F**F***"); //1 - INSIDE
      maskType.add("T*****FF*"); //2 - CONTAINS
      maskType.add("T*F**FFF*"); //3 - EQUAL
      maskType.add("T*****FF*"); //4 - COVERS
      maskType.add("*T****FF*"); //5 - COVERS
      maskType.add("***T**FF*"); //6 - COVERS
      maskType.add("****T*FF*"); //7 - COVERS
      maskType.add("T*F**F***"); //8 - COVEREDBY
      maskType.add("*TF**F***"); //9 - COVEREDBY
      maskType.add("**FT*F***"); //10 - COVEREDBY
      maskType.add("**F*TF***"); //11 - COVEREDBY
      maskType.add("T*T***T**"); //12 - OVERLAP
      maskType.add("1*T***T**"); //13 - OVERLAP

      for (int i=0; i < maskType.size(); i++){
            bboxFilter.add("st_relate(co.ENVELOPE,st_geomFromText(
            'POLYGON((" + minX + " " + minY + "," + maxX + " " + minY +
            "," + maxX + " " + maxY + "," + minX + " " + maxY + "," + minX
            + " " + minY + "))'," + bboxSrid + "), '" + maskType.get(i) +
                  "') = 'TRUE'");
      }
}
else
{
      maskType.add("T*F**F***"); //1 - INSIDE
      maskType.add("T*F**F***"); //2 - COVEREDBY
      maskType.add("*TF**F***"); //3 - COVEREDBY
      maskType.add("**FT*F***"); //4 - COVEREDBY
      maskType.add("**F*TF***"); //5 - COVEREDBY
      maskType.add("T*F**FFF*"); //6 - EQUAL

      for (int i=0; i < maskType.size(); i++){
            bboxFilter.add("st_relate(co.ENVELOPE,st_geomFromText(
            'POLYGON((" + minX + " " + minY + "," + maxX + " " + minY +
            "," + maxX + " " + maxY + "," + minX + " " + maxY + "," + minX
            + " " + minY + "))'," + bboxSrid + "), '" + maskType.get(i) +
                  "') = 'TRUE'");
      }
}

318   // if (bboxFilter != null)
wdh   //    query.append(bboxFilter).append(" and ");
      if (bboxFilter != null){
            for(int i=0; i<bboxFilter.size(); i++){
                  query.append(bboxFilter.get(i)).append(" or ");
            }
            query.substring(0, query.length() - 4);
            query.append(" and ");
      }
```

3c. Create Table ohne „nologging“

Es gibt keine „nologging“-Option für CREATE-Statements in PostgreSQL, daher ausklammern.

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

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

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

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

3d. Abfrage-Statements für den Export

Die Anfragen an die Datenbank aus den Importer-Klassen, verwenden teilweise Oracle-spezifische Funktionen auf Spalten.

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

```
138   // nvl(sd.TEX_IMAGE.getContentLength(), 0) as DB_TEX_IMAGE_SIZE,
wdh   // sd.TEX_IMAGE.getMimeType() as DB_TEX_IMAGE_MIME_TYPE, sd.TEX_MIME_TYPE,
      COALESCE(length(sd.TEX_IMAGE), 0) as DB_TEX_IMAGE_SIZE, sd.TEX_MIME_TYPE,
```

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

```
134   // SYSDATE
      now()
```

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

```
63    // CITYOBJECT_GENERICATT_SEQ.nextval
      nextval('CITYOBJECT_GENERICATTRIB_ID_SEQ')
```

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

```
58    // EXTERNAL_REF_SEQ.nextval
      nextval('EXTERNAL_REFERENCE_ID_SEQ')
```

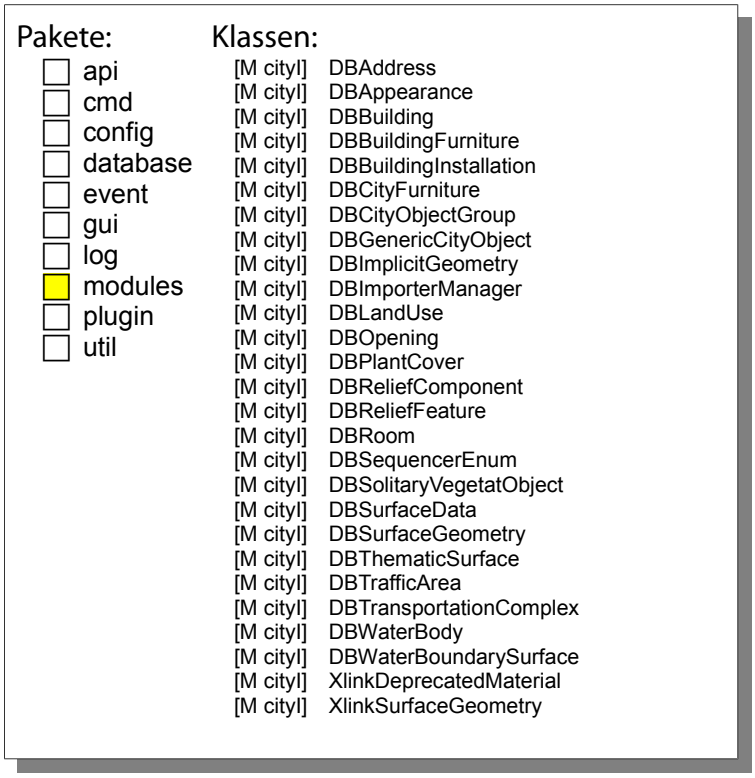
de.tub.citydb.modules.citygml.importer.database.content.**DBSequencer**

```
53    // pstsmt = conn.prepareStatement("select " + sequence.toString() +
            ".nextval from dual");
      pstsmt = conn.prepareStatement("select nextval('" + sequence.toString() +
            "')");
```

de.tub.citydb.modules.citygml.importer.database.content.**DBSurfaceGeometry**
de.tub.citydb.modules.citygml.importer.database.xlink.resolver.**XlinkSurfaceGeometry**

```
126   // SURFACE_GEOMETRY_SEQ.nextval
/98   nextval('SURFACE_GEOMETRY_ID_SEQ')
```

4. Generierte Sequenzen

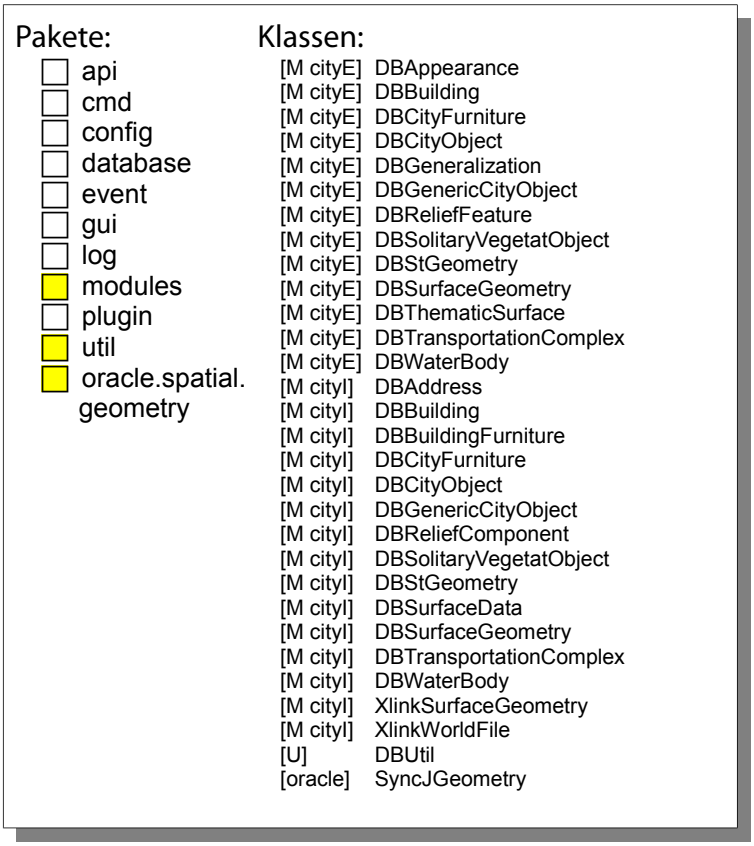


Die ID-Spalten der Tabellen, die als Primary Keys gesetzt sind, wurden in PostgreSQL mit dem Datentyp SERIAL definiert. Dieser Datentyp erstellt implizit ein Sequenz, deren Namen sich aus dem Tabellen- und Spaltennamen sowie der Endung SEQ zusammensetzt. Der Befehl „CREATE SEQUENCE“ muss nicht manuell geschrieben werden, die SQL-Deklaration wird aber in gleicher Form in der Datenbank abgespeichert wie es bei Oracle der Fall ist. In den Skripten für die Oracle-Fassung wurden die Namen der Sequenzen teilweise abgekürzt und beinhalten nicht das Kürzel „ID“ für den Spaltennamen. Das muss im Code berücksichtigt werden, wie auch 3d im vorherigen Kapitel anschaulich zeigte. Darüber hinaus muss die Klasse DBSequencerEnum und all ihre Aufrufe angepasst werden. Alle betroffenen Klassen sind der Box zu entnehmen. Die ID-Umsetzungen innerhalb von Select-Statements aus 3d werden hier nicht nochmal berücksichtigt.

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

```
32    //public enum DBSequencerEnum {
      //    ADDRESS_SEQ,
      //    APPEARANCE_SEQ,
      //    CITYOBJECT_SEQ,
      //    SURFACE_GEOMETRY_SEQ,
      //    IMPLICIT_GEOMETRY_SEQ,
      //    SURFACE_DATA_SEQ,
      //}
      public enum DBSequencerEnum {
            ADDRESS_ID_SEQ,
            APPEARANCE_ID_SEQ,
            CITYOBJECT_ID_SEQ,
            SURFACE_GEOMETRY_ID_SEQ,
            IMPLICIT_GEOMETRY_ID_SEQ,
            SURFACE_DATA_ID_SEQ,
      }
```

5. Die Verarbeitung von Geometrien



Die Anpassungen der Geometrien stellte ohne Zweifel die größte Herausforderung bei der Portierung des Importer-Exporters dar. Es folgt eine kurze Beschreibung, wie man die Geometrien von CityGML in eine Oracle-Datenbank und wie eine PostGIS-Datenbank abbildet und wieder zurück zum Ausgangsformat gelangt.

5a. Von CityGML zur Datenbank

Im JDBC für Orache Spatial sind die geometrien Operationen in der Klasse JGeometry konzentriert. Eine Instanz von JGeometry repräsentiert Oracle-Spatial-Geometrien in Java. Für alle Geometrietypen gibt es entsprechende Methoden, die JGeometry zurück geben. Sie benötigen stets ein Array mit den Koordinaten, eine Dimensionsangabe sowie den Wert des SRIDs. Die Geometrie-Daten aus CityGML werden zuerst in Listen abgelegt, müssen dann aber in Arrays gespeichert werden, um von den beschriebenen Methoden verwendet werden zu können. Bevor die Geometrie an die Datenbank übergeben wird, muss JGeometry in einen strukturierten Datentyp umgewandelt werden, um von einer Statement-Methode als Java.Object verstanden zu werden. Dafür gibt es auch einen Oracle-eigenen Datentyp namens STRUCT. STRUCT representiert quasi den Datentyp SDO_GEOMETRY in seiner Datenbank-Form in Java.

Bei dem JDBC von PostGIS wird mit zwei Klassen gearbeitet. Geometry representiert die PostGIS-Geometrien in Java und umfasst zusammen mit seinen Unterklassen etliche geometrische Operationen. Um Instanzen zu erstellen, muss hingegen eine geomFromString-Methode von PGgeometry verwendet werden. Spezifische Create-Methoden für unterschiedlich Geometrie-Typen gibt es nicht. Um die CityGML-Geometrien in einen String abzuleiten, wurde einfach durch die bereits erwähnten Listen iteriert und der String Stück für Stück zusammen gesetzt. Anders als bei SDO_GEOMTERY wird nicht zwischen jeder Ordinate ein Komma gesetzt, sondern erst nach jedem Punkt. Zwischen x-, y- und ggf. z-Wert

befindet sich, getreu dem WKT-Format, ein Leerzeichen. Zusammen mit der SRID-Information ist der String vollständig und kann der Methode übergeben werden. Wie schon bei Oracle muss die Geometrie in ein Format umgewandelt werden, was von der Datenbank interpretiert werden kann, in diesem Fall eine Instanz von PGgeometry. Die geht mit einem einfach Konstruktor `PGgeometry(Geometry)`.

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

```
91    // private DBSdoGeometry sdoGeometry;
wdh+  private DBStGeometry stGeometry;

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

133   // JGeometry multiPoint = null;
wdh+  PGgeometry multiPoint = null;

224   // multiPoint = sdoGeometry.getMultiPoint(address.getMultiPoint());
wdh+  multiPoint = stGeometry.getMultiPoint(address.getMultiPoint());

      // if (multiPoint != null) {
      //    Struct multiPointObj= SyncJGeometry.syncStore(multiPoint,batchConn);
      //    psAddress.setObject(8, multiPointObj);
      // } else
      //    psAddress.setNull(8, Types.STRUCT, "MDSYS.SDO_GEOMETRY");
      if (multiPoint != null) {
            psAddress.setObject(8, multiPoint);
      } else
            psAddress.setNull(8, Types.OTHER, "ST_GEOMETRY");
```

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

```
211   // double[] ordinates = new double[points.size()];
wdh+  // int i = 0;
      // for (Double point : points)
      //    ordinates[i++] = point.doubleValue();
      // JGeometry boundedBy =
      //    JGeometry.createLinearPolygon(ordinates, 3, dbSrid);
      // STRUCT obj = SyncJGeometry.syncStore(boundedBy, batchConn);
      //
      // psCityObject.setObject(4, obj);
      String geomEWKT = "SRID=" + dbSrid + ";POLYGON((";
      for (int i=0; i<points.size(); i+=3){
            geomEWKT += points.get(i) + " " + points.get(i+1) + " " +
                  points.get(i+2) + ",";
      }
      geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
      geomEWKT += "))";

      Geometry boundedBy = PGgeometry.geomFromString(geomEWKT);
      PGgeometry pgBoundedBy = new PGgeometry(boundedBy);

      psCityObject.setObject(4, pgBoundedBy);
```

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

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

```
88    // public JGeometry getPoint(PointProperty pointProperty) {
wdh   //    JGeometry pointGeom = null;
      public PGgeometry getPoint(PointProperty pointProperty) throws
      SQLException {
            Geometry pointGeom = null;

99    // double[] coords = new double[values.size()];
      // int i = 0;
      // for (Double value : values)
      //    coords[i++] = value.doubleValue();
      // pointGeom = JGeometry.createPoint(coords, 3, dbSrid);
      pointGeom = PGgeometry.geomFromString("SRID=" + dbSrid + ";POINT(" +
            values.get(0) + " " + values.get(1) + " " + values.get(2) + ")");

171   // if (!pointList.isEmpty()) {
wdh   //    Object[] pointArray = new Object[pointList.size()];
      //    int i = 0;
      //    for (List<Double> coordsList : pointList) {
      //          if (affineTransformation)
      //                dbImporterManager.getAffineTransformer().
      //                      transformCoordinates(coordsList);
      //
      //          double[] coords = new double[3];
      //
      //          coords[0] = coordsList.get(0).doubleValue();
      //          coords[1] = coordsList.get(1).doubleValue();
      //          coords[2] = coordsList.get(2).doubleValue();
      //
      //          pointArray[i++] = coords;
      //    }
      //    multiPointGeom = JGeometry.createMultiPoint(pointArray, 3, dbSrid);
      // }
      // }
      // return multiPointGeom;
      if (!pointList.isEmpty()) {
            String geomEWKT = "SRID=" + dbSrid + ";MULTIPOINT(";

            for (List<Double> coordsList : pointList){

                  if (affineTransformation)
                        dbImporterManager.getAffineTransformer().
                              transformCoordinates(coordsList);

                  geomEWKT += coordsList.get(0) + " " + coordsList.get(1) + " "
                        + coordsList.get(2) + ",";
            }

            geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
            geomEWKT += ")";

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

213   // if (!pointList.isEmpty()) {
wdh   //    Object[] pointArray = new Object[pointList.size()];
      //    int i = 0;
      //    for (List<Double> coordsList : pointList) {
```

```
//          if (affineTransformation)
//                dbImporterManager.getAffineTransformer().
//                      transformCoordinates(coordsList);
//          double[] coords = new double[coordsList.size()];
//          int j = 0;
//          for (Double coord : coordsList)
//                coords[j++] = coord.doubleValue();
//
//          pointArray[i++] = coords;
//    }
//    multiCurveGeom = JGeometry.createLinearMultiLineString(pointArray,
//    3, dbSrid);
// }
if (!pointList.isEmpty()) {
      String geomEWKT = "SRID=" + dbSrid + ";MULTILINESTRING((";

      for (List<Double> coordsList : pointList) {
            if (affineTransformation)
                  dbImporterManager.getAffineTransformer().
                        transformCoordinates(coordsList);

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

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

```
437   // JGeometry geom = new JGeometry(coords.get(0), coords.get(1), dbSrid);
      // STRUCT obj = SyncJGeometry.syncStore(geom, batchConn);
      // psSurfaceData.setObject(15, obj);
      Geometry geom = PGgeometry.geomFromString("SRID=" + dbSrid + ";POINT(" +
            coords.get(0) + " " + coords.get(1) + ")");
      PGgeometry pgGeom = new PGgeometry(geom);
      psSurfaceData.setObject(15, pgGeom);
```

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

```
281   // 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((";
      ComposedGeom polyGeom = (ComposedGeom)geomNode.geometry;
      int dimensions = geomNode.geometry.getDimension();

      for (int i = 0; i < polyGeom.numGeoms(); i++){
      if (dimensions == 2)
            for (int j=0; j<polyGeom.getSubGeometry(i).numPoints(); j++){
                  geomEWKT += polyGeom.getSubGeometry(i).getPoint(j).x + "
                  " + polyGeom.getSubGeometry(i).getPoint(j).y + ",";
            }

      if (dimensions == 3)
            for (int j=0; j<polyGeom.getSubGeometry(i).numPoints(); j++){
                  geomEWKT += polyGeom.getSubGeometry(i).getPoint(j).x + "
                  " + polyGeom.getSubGeometry(i).getPoint(j).y + " " +
                  polyGeom.getSubGeometry(i).getPoint(j).z + ",";
            }

            geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
            geomEWKT += "),(";
      }

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

      Geometry geom = PGgeometry.geomFromString(geomEWKT);
      geomNode.geometry = geom;
}

382   // protected JGeometry geometry;
wdh+  protected Geometry geometry;
```

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

```
      // 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);
```

5b. Von der Datenbank zurück zu CityGML

Einfach gesagt, funktioniert der Exportweg genau anders herum. In Oracle wird ein ResultSet zuerst in den STRUCT-Datentyp gecastet und dann mit JGeometry.load(struct) als Java-Geometrie erstellt. Das gleiche gilt bei PostGIS für PGgeometry und der Methode getGeometry. Mit den Methoden von JGeometry lassen sich die Information der Geometrien bequem in Arrays speichern und für die CityGML-Primitive in Listen prozessieren. Bei PostGIS reichen die Methoden der Geometry-Klasse alleine nicht aus. Es müssen Unterklassen wie ComposedGeom oder MultiLineString eingesetzt werden, um Sub-Geometrien anzusprechen, mit deren Werte dann die Listen befüllt werden. Glücklicherweise führte dies zu keinen Namenskonflikte mit den CityGML-Primitiven.

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

```
822   // STRUCT struct = (STRUCT)rs.getObject("GT_REFERENCE_POINT");
wdh+  // if (!rs.wasNull() && struct != null) {
      //    JGeometry jGeom = JGeometry.load(struct);
      //    double[] point = jGeom.getPoint();
      //
      //    if (point != null && point.length >= 2) {
      //          Point referencePoint = new PointImpl();
      //          List<Double> value = new ArrayList<Double>();
      //                value.add(point[0]);
      //                value.add(point[1]);
      PGgeometry pgGeom = (PGgeometry)rs.getObject("GT_REFERENCE_POINT");
      if (!rs.wasNull() && pgGeom != null) {
            Geometry geom = pgGeom.getGeometry();
            Point referencePoint = new PointImpl();
                  List<Double> value = new ArrayList<Double>();
                        value.add(geom.getPoint(0).getX());
                        value.add(geom.getPoint(0).getY());
```

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

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

170   // if (geom.getDimension() == 2) {
      //    lower = new Point(points[0], points[1], 0);
      //    upper = new Point(points[2], points[3], 0);
      // } else {
      //    lower = new Point(points[0], points[1], points[2]);
      //    upper = new Point(points[3], points[4], points[5]);
      if (geom.getDimension() == 2) {
            lower = new Point(geom.getFirstPoint().x, geom.getFirstPoint().y,0);
            upper = new Point(geom.getPoint(2).x, geom.getPoint(2).y, 0);
      } else {
            lower = new Point(geom.getFirstPoint().x, geom.getFirstPoint().y,
                  geom.getFirstPoint().z);
            upper = new Point(geom.getPoint(2).x, geom.getPoint(2).y,
                  geom.getPoint(2).z);
```

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

```
121   // double[] points = geom.getOrdinatesArray();
```

```
      // Point lower = new Point(points[0], points[1], points[2]);
      // Point upper = new Point(points[3], points[4], points[5]);
      Point lower = new Point(geom.getFirstPoint().x, geom.getFirstPoint().y,
            geom.getFirstPoint().z);
      Point upper = new Point(geom.getPoint(2).x, geom.getPoint(2).y,
            geom.getPoint(2).z);
```

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

```
94    // public PointProperty getPoint(JGeometry geom, boolean setSrsName) {
      //    PointProperty pointProperty = null;
      //    if (geom != null && geom.getType() == JGeometry.GTYPE_POINT) {
      //          pointProperty = new PointPropertyImpl();
      //          int dimensions = geom.getDimensions();
      //
      //          double[] pointCoord = geom.getPoint();
      //
      //          if (pointCoord != null && pointCoord.length >= dimensions) {
      //                Point point = new PointImpl();
      //
      //                List<Double> value = new ArrayList<Double>();
      //                for (int i = 0; i < dimensions; i++)
      //                      value.add(pointCoord[i]);
      public PointProperty getPoint(Geometry geom, boolean setSrsName) {
            PointProperty pointProperty = null;

            if (geom != null && geom.getType() == 1) {
                  pointProperty = new PointPropertyImpl();
                  int dimensions = geom.getDimension();

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

                        List<Double> value = new ArrayList<Double>();
                        value.add(geom.getPoint(0).getX());
                        value.add(geom.getPoint(0).getY());
                        .
                        .
                  if (dimensions == 3) {
                        Point point = new PointImpl();
                        List<Double> value = new ArrayList<Double>();
                        value.add(geom.getPoint(0).getX());
                        value.add(geom.getPoint(0).getY());
                        value.add(geom.getPoint(0).getZ());

140   // public PolygonProperty getPolygon(JGeometry geom, boolean setSrsName) {
      //    PolygonProperty polygonProperty = null;
      //
      //    if (geom != null && geom.getType() == JGeometry.GTYPE_POLYGON) {
      //          polygonProperty = new PolygonPropertyImpl();
      //          Polygon polygon = new PolygonImpl();
      //          int dimensions = geom.getDimensions();
      //
      //          int[] elemInfoArray = geom.getElemInfo();
      //          double[] ordinatesArray = geom.getOrdinatesArray();
      //
      //          if (elemInfoArray.length < 3 || ordinatesArray.length == 0)
      //                return null;
      //
      //          List<Integer> ringLimits = new ArrayList<Integer>();
      //          for (int i = 3; i < elemInfoArray.length; i += 3)
      //                ringLimits.add(elemInfoArray[i] - 1);
```

```
//
//          ringLimits.add(ordinatesArray.length);
//
//          boolean isExterior = elemInfoArray[1] == 1003;
//          int ringElem = 0;
//          for (Integer curveLimit : ringLimits) {
//                List<Double> values = new ArrayList<Double>();
//
//                for ( ; ringElem < curveLimit; ringElem++)
//                      values.add(ordinatesArray[ringElem]);
//
//                if (isExterior) {
public PolygonProperty getPolygon(Geometry geom, boolean setSrsName) {
      PolygonProperty polygonProperty = null;

      if (geom != null && geom.getType() == 3) {
            polygonProperty = new PolygonPropertyImpl();
            Polygon polygon = new PolygonImpl();
            int dimensions = geom.getDimension();

            if (geom.getValue() == null)
                  return null;

            ComposedGeom polyGeom = (ComposedGeom)geom;

            for (int i = 0; i < polyGeom.numGeoms(); i++){
                  List<Double> values = new ArrayList<Double>();

            if (dimensions == 2)
            for (int j=0; j<polyGeom.getSubGeometry(i).numPoints(); j++){
                  values.add(polyGeom.getSubGeometry(i).getPoint(j).x);
                  values.add(polyGeom.getSubGeometry(i).getPoint(j).y);
            }

            if (dimensions == 3)
            for (int j=0; j<polyGeom.getSubGeometry(i).numPoints(); j++){
                  values.add(polyGeom.getSubGeometry(i).getPoint(j).x);
                  values.add(polyGeom.getSubGeometry(i).getPoint(j).y);
                  values.add(polyGeom.getSubGeometry(i).getPoint(j).z);
            }
            //isExterior
            if (i == 0) {

208 // public MultiPointProperty getMultiPointProperty(JGeometry geom, boolean
wdh // setSrsName) {
//    MultiPointProperty multiPointProperty = null;
//
//    if (geom != null) {
//          multiPointProperty = new MultiPointPropertyImpl();
//          MultiPoint multiPoint = new MultiPointImpl();
//          int dimensions = geom.getDimensions();
//
//    if (geom.getType() == JGeometry.GTYPE_MULTIPOINT) {
//          double[] ordinates = geom.getOrdinatesArray();
//
//          for (int i = 0; i < ordinates.length; i += dimensions) {
//                Point point = new PointImpl();
//
//          List<Double> value = new ArrayList<Double>();
//
//          for (int j = 0; j < dimensions; j++)
//                value.add(ordinates[i + j]);
//                .
//                .
```

```
//          }
//    } else if (geom.getType() == JGeometry.GTYPE_POINT) {
//          .
//          .
public MultiPointProperty getMultiPointProperty(Geometry geom, boolean
setSrsName) {
      MultiPointProperty multiPointProperty = null;

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

      if (geom.getType() == 4) {
            List<Double> value = new ArrayList<Double>();
            Point point = new PointImpl();

            if (dimensions == 2)
                  for (int i = 0; i < geom.numPoints(); i++) {
                        value.add(geom.getPoint(i).x);
                        value.add(geom.getPoint(i).y);
                  }
            if (dimensions == 3)
                  for (int i = 0; i < geom.numPoints(); i++) {
                        value.add(geom.getPoint(i).x);
                        value.add(geom.getPoint(i).y);
                        value.add(geom.getPoint(i).z);
                  }
            .
            .
            }
      }
      else if (geom.getType() == 1) {
            Point point = new PointImpl();
            List<Double> value = new ArrayList<Double>();

            value.add(geom.getPoint(0).x);
            value.add(geom.getPoint(0).y);

            if (dimensions == 3)
                  value.add(geom.getPoint(0).z);

355 // public MultiCurveProperty getMultiCurveProperty(JGeometry geom, boolean
wdh // setSrsName) {
//    MultiCurveProperty multiCurveProperty = null;
//
//    if (geom != null) {
//          multiCurveProperty = new MultiCurvePropertyImpl();
//          MultiCurve multiCurve = new MultiCurveImpl();
//          int dimensions = geom.getDimensions();
//
//          if (geom.getType() == JGeometry.GTYPE_MULTICURVE ) {
//                int[] elemInfoArray = geom.getElemInfo();
//                double[] ordinatesArray = geom.getOrdinatesArray();
//
//                if (elemInfoArray.length < 3 ||
//                      ordinatesArray.length == 0)
//                            return null;
//
//                List<Integer> curveLimits = new ArrayList<Integer>();
//                      for (int i = 3; i < elemInfoArray.length; i += 3)
//                            curveLimits.add(elemInfoArray[i] - 1);
//
//                curveLimits.add(ordinatesArray.length);
```

```
//
//                int curveElem = 0;
//                for (Integer curveLimit : curveLimits) {
//                      List<Double> values = new ArrayList<Double>();
//
//                      for ( ; curveElem < curveLimit; curveElem++)
//                            values.add(ordinatesArray[curveElem]);
//                      .
//                      .
//                      curveElem = curveLimit;
//                }
//          }
//          else if (geom.getType() == JGeometry.GTYPE_CURVE ) {
//                double[] ordinatesArray = geom.getOrdinatesArray();
//                List<Double> value = new ArrayList<Double>();
//
//                for (int i = 0; i < ordinatesArray.length; i++)
//                      value.add(ordinatesArray[i]);
public MultiCurveProperty getMultiCurveProperty(Geometry geom, boolean
setSrsName) {
      MultiCurveProperty multiCurveProperty = null;

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

      if (geom.getType() == 5) {
            MultiLineString mlineGeom = (MultiLineString)geom;

            for (int i = 0; i < mlineGeom.numLines(); i++){
                  List<Double> values = new ArrayList<Double>();

                  if (dimensions == 2)
                        for (int j=0; j<mlineGeom.getLine(i).numPoints();
                        j++){
                          values.add(mlineGeom.getLine(i).getPoint(j).x);
                          values.add(mlineGeom.getLine(i).getPoint(j).y);
                        }
                  if (dimensions == 3)
                        for (int j=0; j<mlineGeom.getLine(i).numPoints();
                        j++){
                          values.add(mlineGeom.getLine(i).getPoint(j).x);
                          values.add(mlineGeom.getLine(i).getPoint(j).y);
                          values.add(mlineGeom.getLine(i).getPoint(j).z);
                  }
                  .
                  .
            }
      }
      else if (geom.getType() == 2) {
            List<Double> value = new ArrayList<Double>();

            if (dimensions == 2)
                  for (int i = 0; i < geom.numPoints(); i++){
                        value.add(geom.getPoint(i).x);
                        value.add(geom.getPoint(i).y);
                  }
            if (dimensions == 3)
                  for (int i = 0; i < geom.numPoints(); i++){
                        value.add(geom.getPoint(i).x);
                        value.add(geom.getPoint(i).y);
                        value.add(geom.getPoint(i).z);
                  }
```

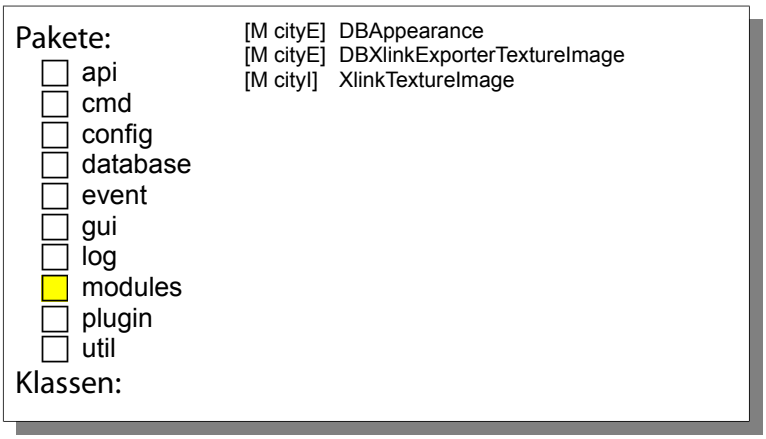
de.tub.citydb.util.database.**DBUtil**

```
309   // STRUCT struct = (STRUCT)rs.getObject(1);
wdh+  // if (!rs.wasNull() && struct != null) {
      //    JGeometry jGeom = JGeometry.load(struct);
      //    int dim = jGeom.getDimensions();
      PGgeometry pgGeom = (PGgeometry)rs.getObject(1);
      if (!rs.wasNull() && pgGeom != null) {
            Geometry geom = pgGeom.getGeometry();
            int dim = geom.getDimension();
```

5c. Synchronisation von Geometriefunktionen

Es ist leider erwiesen, dass die für diese Umwandlung benötigte Methode store(JGeometry) nicht threadsafe ist. Das bedeutet, wenn mehrere Prozesse die gleiche Geometrie umwandeln wollen, kann es passieren, dass Deadlocks entstehen, also eine Prozess auf den jeweils andere wartet, so dass nichts mehr passiert. Dafür existiert die Methode SyncJGeometry, die bei jeder STRUCT-Umwandlung aufgerufen wird und die Prozesse in eine Warteschleife zwingt und damit synchronisiert. Bisher ist nicht bekannt, ob ein ähnliches Problem auch bei den JDBC-Klassen von PostGIS auftritt. Sollte dies nicht der Fall sein, könnte das zu einem erheblich Performance-Vorteil führen.

6. Die Verarbeitung von Texturen



Der Im- und Export von Texturen musste deutlich anders gestaltet werden. Das liegt an dem Oracle-eigenen Datentyp ORDIMAGE. Oft werden Bild-Dateien als Binary Large Objects (BLOB) in eine Datenbank gespeichert. So auch bei PostgreSQL (bytea = Byte Array). ORDIMAGE hat den Vorteil, das Metadaten zu einem Bild abgerufen werden können und das einige Bildbearbeitungs-Methoden mitgeliefert werden. Diese werden u.a. in der Klasse DBAppearance aufgerufen (siehe Kapitel 3d). Bei BLOBs ist der Funktionsumfang sehr eingeschränkt. Da aber in der 3DCityDB bisher kaum von den Möglichkeiten von ORDIMAGE Gebrauch gemacht wird, ja sogar Oracle selbst die Verwendung von BLOBs empfielt, ergab sich kein Nachteil für die PostGIS-Umsetzung.

6a. Der Import von Texturen

Wie man an den folgenden Quellcode-Beispielen sehen kann, reduziert sich das Importieren von Texturen auf wenige Zeilen, vor allem weil ORDIMAGE vor dem Einladen von Bildern noch mit einem Datenbank-Statement initialisiert werden muss. Für PostgreSQL reichen InputStreams.

```
74    // psPrepare = externalFileConn.prepareStatement(
            "update SURFACE_DATA set TEX_IMAGE=ordimage.init() where ID=?");
      // psSelect = externalFileConn.prepareStatement(
            "select TEX_IMAGE from SURFACE_DATA where ID=? for update");
      // psInsert = (OraclePreparedStatement)externalFileConn.prepareStatement(
            "update SURFACE_DATA set TEX_IMAGE=? where ID=?");
      psInsert = externalFileConn.prepareStatement(
            "update SURFACE_DATA set TEX_IMAGE=? where ID=?");

113+  // // second step: prepare ORDIMAGE
      // psPrepare.setLong(1, xlink.getId());
      // psPrepare.executeUpdate();
      //
      // // third step: get prepared ORDIMAGE to fill it with contents
      // psSelect.setLong(1, xlink.getId());
      // OracleResultSet rs = (OracleResultSet)psSelect.executeQuery();
      //    if (!rs.next()) {
      //          LOG.error("Database error while importing texture file '" +
      //                imageFileName + "'.");
      //
      //          rs.close();
      //          externalFileConn.rollback();
      //          return false;
      //    }

114   // OrdImage imgProxy = (OrdImage)rs.getORAData(
      //    1,OrdImage.getORADataFactory());
      // rs.close();
      FileInputStream fis = new FileInputStream(imageFile);

120   // 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();
      if (isRemote) {
```

```
            InputStream stream = imageURL.openStream();
            psInsert.setBinaryStream(1, stream);
        } else {
            psInsert.setBinaryStream(1, fis, (int)imageFile.length());
        }
        psInsert.setLong(2, xlink.getId());
        psInsert.execute();
        externalFileConn.commit();
        return true;
```

6b. Der Export von Texturen

Für PostgreSQL existieren zwei Möglichkeiten, Bilder aus der Datenbank zurück in Dateien zu schreiben. Es konnte bisher kein zeitlicher Unterschied festgestellt werden. Die erste Variante wird derzeit bevorzugt, weil keine feste Größe für das Byte-Array, welches den Stream auffängt, angegeben werden muss und sie damit dynamischer erscheint.

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

```
126     // OracleResultSet rs = (OracleResultSet)psTextureImage.executeQuery();
        ResultSet rs = (ResultSet)psTextureImage.executeQuery();

141     // // read oracle image data type
        // 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();
        // }
```

1.Variante:

```
        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;
        }
```

2.Variante:

```
        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;
        }
```

7. Die Batchsize von PostgreSQL

Die maximale Batchsize von PostgreSQL wurde in der Internal-Klasse bisher nur pauschal auf 10000 gesetzt. Evtl. ist mehr möglich. In allen in der Box aufgeführten Klassen (auf der nächsten Seite) muss meist nur an einer Stelle der Attributname von `ORACLE` auf *`POSTGRESQL`* gesetzt werden.

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

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

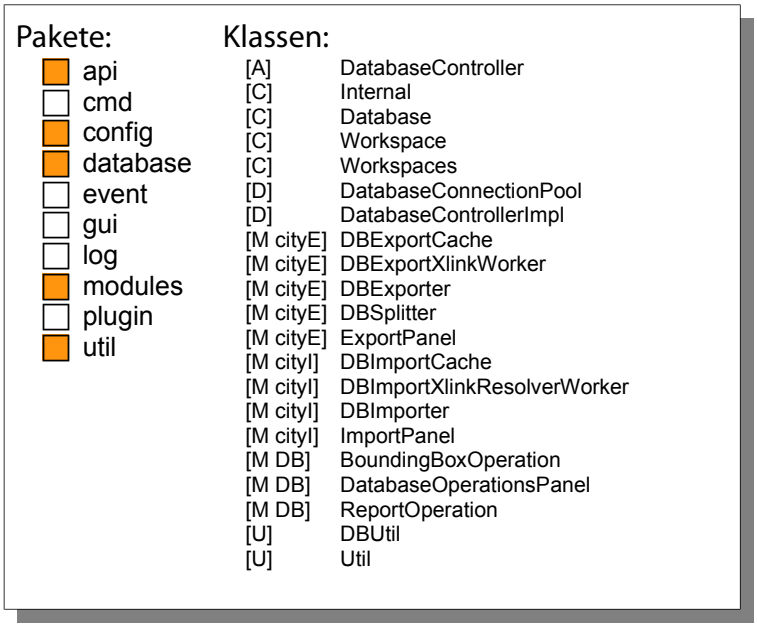
In den folgenden zwei Klassen konnte keine equivalente Methode für das Java-Prepared-Statement gefunden werden. Der Statement-Batch wird mit `executeBatch()` ausgeführt (statt `sendBatch()`).

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

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

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


Pakete:
api
cmd
config
database
event
gui
log
modules
plugin
util
Klassen:
[C] Internal
[C] UpdateBatching
[M cityE] DBExportCache
[M cityI] DBImportXlinkResolverWorker
[M cityI] DBImportXlinkWorker
[M cityI] DBAddress
[M cityI] DBAddressToBuilding
[M cityI] DBAppearance
[M cityI] DBAppearToSurfaceData
[M cityI] DBBuilding
[M cityI] DBBuildingFurniture
[M cityI] DBBuildingInstallation
[M cityI] DBCityFurniture
[M cityI] DBCityObject
[M cityI] DBCityObjectGenericCityObject
[M cityI] DBCityObjectGroup
[M cityI] DBExternalReference
[M cityI] DBGenericCityObject
[M cityI] DBImplicitGeometry
[M cityI] DBLandUse
[M cityI] DBOpening
[M cityI] DBOpeningToThemSurface
[M cityI] DBPlantCover
[M cityI] DBReliefComponent
[M cityI] DBReliefFeatToRelComp
[M cityI] DBReliefFeature
[M cityI] DBRoom
[M cityI] DBSolitaryVegetatObject
[M cityI] DBSurfaceData
[M cityI] DBSurfaceGeometry
[M cityI] DBThematicSurface
[M cityI] DBTrafficArea
[M cityI] DBTransportationComplex
[M cityI] DBWaterBodyToWaterBndSrf
[M cityI] DBWaterBody
[M cityI] DBWaterBoundarySurface
[M cityI] DBImportCache
[M cityI] DBXlinkImporterBasic
[M cityI] DBXlinkImporterDeprecatedMaterial
[M cityI] DBXlinkImporterGroupToCityObject
[M cityI] DBXlinkImporterLibraryObject
[M cityI] DBXlinkImporterLinearRing
[M cityI] DBXlinkImporterSurfacegeometry
[M cityI] DBXlinkImporterTextureAssociation
[M cityI] DBXlinkImporterTextureFile
[M cityI] DBXlinkImporterTextureParam
[M cityI] XlinkBasic
[M cityI] XlinkDeprecatedMaterial
[M cityI] XlinkGroupToCityObject
[M cityI] XlinkSurfaceGeometry
[M cityI] XlinkTexCoordList
[M cityI] XlinkTextureAssociation
[M cityI] XlinkTextureParam
[M cityI] XlinkWorldFile
[M cityI] ResourcesPanel

8. Workspace- und Versionierungs-Management



Ein Workspace- und Versionierungs-Management wie für Oracle Spatial, ist in PostgreSQL / PostGIS nicht enthalten. Jegliche Schnittstellen im Quellcode mussten daher ausgeklammert werden (sind aber nicht gelöscht worden). In der Übersichtsbox kann man nachvollziehen, welche Klassen verändert werden müssten, sollte in Zukunft eine Workspace-Lösung für PostgreSQL entwickelt werden.