Port of the 3D-CityDB from Oracle Spatial to PostGIS

Changes on the Importer/Exporter-Tool

(by Felix Kunde)

Content:

0. Legend

The Boxes at the start of each chapter should give a quick overview which classes had to be changed and which packages were affected by this.

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 neccessary 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 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:	Classes:	
api	[Cmd]	ImpExpCmd
cmd	[C]	DBConnection
config	[D]	DatabaseConnectionPool
database	[D]	DatabaseControllerImpl
event	[M cityC]	BranchTemporaryCacheTable
gui	[M cityC]	CacheManager
log	[M cityC]	HeapCacheTable
modules	[M cityC]	TemporaryCacheTable
plugin	[M cityE]	DBExportWorker
util	[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]	IlegalPluginEventChecker
	[U]	DBUtil

Connection-handling hasn't changed much for the *PostgreSQL*-database only because the *Universal Connection Pool (UCP)* by Oracle is still used. The `PoolDataSource` of the *UCP* must pool a proper DataSource of *PostgreSQL* (`PGSimpleDataSource`). It was necessary to set the database-name separately. The method `conn.getSid()` fetches the right value of the according text-field but can't interpret it internally. Obviously that's because of the different definitions about the database itself between *Oracle* and *PostgreSQL*. To work within a network the server-name and the port-number have to be set as well. The URL which usually addresses the JDBC-Driver of a DBMS, could be left out. Connection-properties were uncommented as the connection-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 *3D-CityDB*. The Apache *Jakarta DBCP* was tested by the developers but found to work unacceptably worse than the *UCP*. The Connection Pools of Apache's *Tomcat 7* or *C3PO* should be an alternative. As seen by the number of orange packages in the overview-box, this means a lot of code-rework.

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

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

180   // für KML-Export (maybe this doesn't need to be changed)
```

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

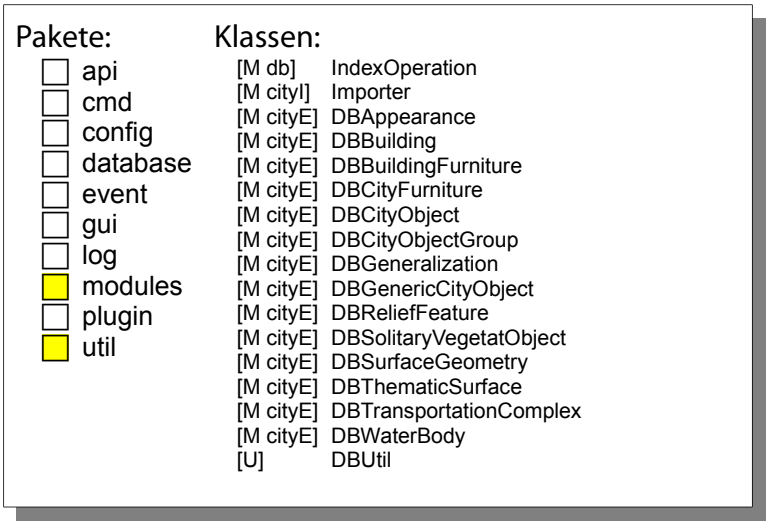
de.tub.citydb.database.**DatabaseConnectionPool**

```
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());
```

or:

```
      poolDataSource.setServerName(conn.getServer());
      poolDataSource.setPortNumber(conn.getPort());

115+  // set connection properties
```

2. Calling the PL/pgSQL-functions



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. Within Java only the names of the called functions were changed.

2a. index-functions, database-report, utility-functions inside of statements

The bigger the size of the files to be imported the longer it takes when data is indexed after every inserted row. Therefore indexes are dropped and recreated after the Import. *Oracle* keeps Metadata of a dropped index, *PostgreSQL* doesn't. An alternative way was programmed but it's 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. Performance was only tested with small datasets. If the switch-case is used in the future, classes `IndexOperation` and `Importer` need to be changed. Corresponding PL/pgSQL-Scripts have to be added as well, indeed. They are already written but are not a part of the recent release.

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

Drop-Case

```
301   if (!parts[4].equals("DROPPED")) {
```

rep+ Switch-Case

```
      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
rep   //    geodb_stat.table_contents}");
      callableStmt = (CallableStatement)conn.prepareCall("{? = call
            geodb_pkg.stat_table_contents()}");

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

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

~400  // 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);

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

2b. Calculation of the BoundingBox

For the calculation of 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't be shorten in number of points like in *Oracle* (LLB, URT), 5 Points were needed for the coordinate-transformation. The query did not work with a `PreparedStatement`. Thus a `statement`-object was executed directly.

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);

625   // 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());
      conn = dbConnectionPool.getConnection();
      query = conn.createStatement();
```

```
rs = query.executeQuery("select ST_TRANSFORM(ST_GeomFromText('POLYGON((" +
      bbox.getLowerLeftCorner().getX() + " " +
      bbox.getLowerLeftCorner().getY() + "," +
      bbox.getUpperRightCorner().getX() + " " +
      bbox.getLowerLeftCorner().getY() + "," +
      bbox.getUpperRightCorner().getX() + " " +
      bbox.getUpperRightCorner().getY() + "," +
      bbox.getLowerLeftCorner().getX() + " " +
      bbox.getUpperRightCorner().getY() + "," +
      bbox.getLowerLeftCorner().getX() + " " +
      bbox.getLowerLeftCorner().getY() + "))'," +
      sourceSrid + "),," + targetSrid + ")");

// 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. Statement-Strings and database-SRS

Pakete:	Klassen:	
☐ api	[A]	DatabaseSrsType
☐ cmd	[A]	DatabaseSrs
☐ config	[G]	SrsComboBoxFactory
☐ database	[M cityC]	CacheTableBasic
☐ event	[M cityC]	CacheTableDeprecatedMaterial
☐ gui	[M cityC]	CacheTableGlobalAppearance
☐ log	[M cityC]	CacheTableGmlId
☐ modules	[M cityC]	CacheTableGroupToCityObject
☐ plugin	[M cityC]	CacheTableLiberaryObject
☐ util	[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

3a. The database-SRS

Until now *PostGIS* doesn't offer 3D-spatial-reference-systems by default. Insert-examples for *PostGIS* can be found at spatialreference.org. It's 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.

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%'" :
                  ""); --could not be translated properly
```

3b. 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's possible to combine the mask-attributes with a logical OR (+). This is not adoptable for *PostGIS*, as the equivalent ST_Relate-operation can only use one mask. Thus the query-string is built by iterations through a list with the mask-attributes (`maskTypes`). The StringBuilder is used for building the query-string.

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**

```
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>();
StringBuilder tmp = new StringBuilder();

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
}
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
}

int dbSrid = dbConnectionPool.getActiveConnectionMetaData().
                  getReferenceSystem().getSrid();

String geom = "st_geomFromText('POLYGON((" +
      minX + " " + minY + "," +
      maxX + " " + minY + "," +
      maxX + " " + maxY + "," +
      minX + " " + maxY + "," +
      minX + " " + minY + "))'," + bboxSrid + ")";

// srid of query window cannot be different from database srid
if (bboxSrid != dbSrid)
geom = "geodb_pkg.util_transform_or_null(" + geom + ", " + dbSrid + ")";

tmp.append("(");

for (int i=0; i < maskType.size(); i++){
      tmp.append("st_relate(co.ENVELOPE, " + geom + ", '"
            + maskType.get(i) + "') = 'TRUE'");

      if (i < maskType.size() - 1)
      tmp.append(" or ");
}
tmp.append(")");
bboxFilter = tmp.toString();
```

3c. Query-statements for Import

Some queries of the Importer-classes use *Oracle*-specific functions.

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,
```

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')
```

3d. Create Table without "nologging"

There is no `nologging`-Option for CREATE-statements in *PostgreSQL.*

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

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

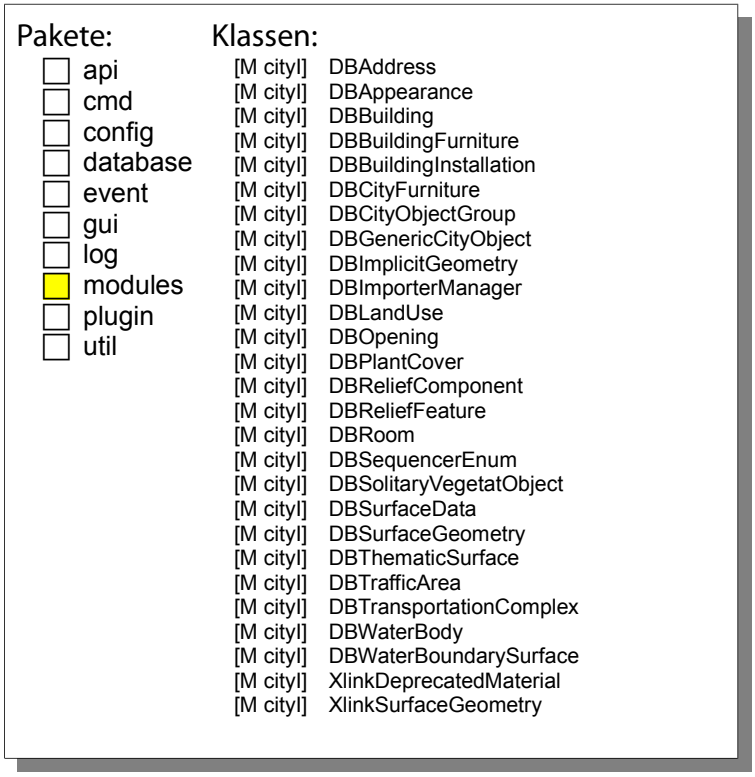
de.tub.citydb.modules.citygml.common.database.cache.**HeapCacheTable**

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

3e. 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

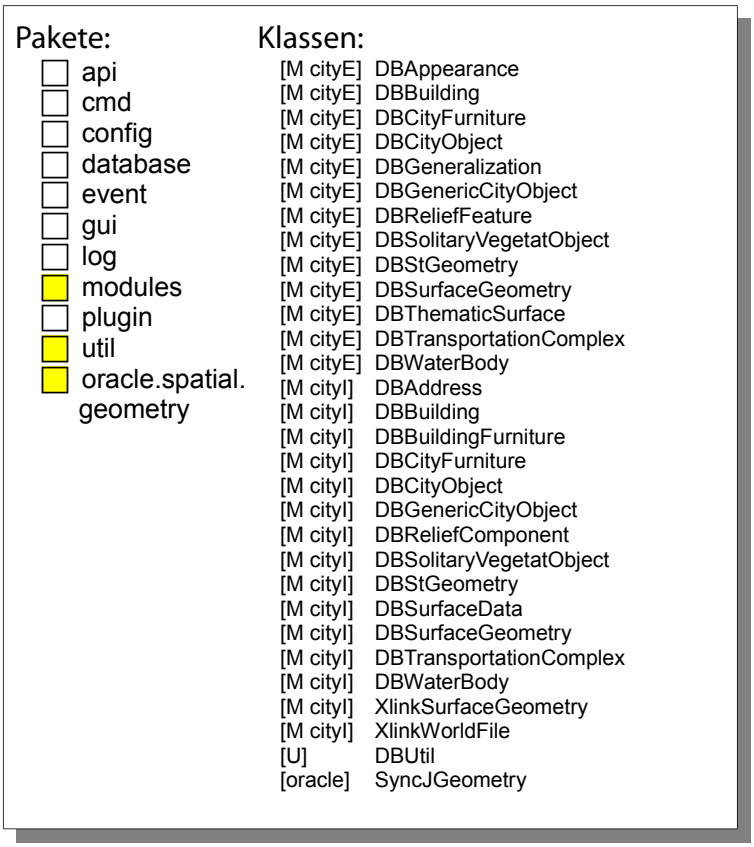


In *PostgreSQL* it's quite common to assign the data type SERIAL to ID-columns which are used as primary keys. SERIAL implicitly creates a sequence with the names of the table, the column and the ending "_SEQ". The declaration "CREATE SEQUENCE" must not be written manually like in Oracle. But this holds a trap. As names are created automatically with SERIAL they differ from the customized names in *Oracle*. See also ***3c*** for examples.

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. How to work with database geometries in Java



Translating the processing of geometries to the *PostGIS*-JDBC 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.

5a. From CityGML to *3D-CityDB*

The *Oracle*-JDBC handles geometries with one central class called `JGeometry`. One instance of `JGeometry` represents SDO_GEOMETRY in Java. All methods of the different geometric types return `JGeometry`. They need an array of coordinates, the number of dimensions and a known SRID for doing so. The geometries of CityGML are described by geometric primitives from the `citygml4j.lib`. Their values are first transferred to list-elements and then iterated into arrays to be used by the described `JGeometry`-methods. Unfortunately `JGeometry` can't be set as an object for the database-statements. It needs to be stored into a `STRUCT`-object. `STRUCT` fits better to the database-structure of SDO_GEOMETRY.

The PostGIS-JDBC works with two geometry-classes – `Geometry` and `PGgeometry`. They can be slightly compared to `JGeometry` and `STRUCT`, but there are greater difference within the methods. `Geometry` offers some geometric operations, but to create an instance of `Geometry` the `PGgeometry`-method `geomFromString(String)` has to be used. So the values of list-elements have to iteratively build up a string and not fill an array. The String is representing 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 constructed as a `PGgeometry`-object and then passed to the `PreparedStatement`.

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

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

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

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

224    // multiPoint = sdoGeometry.getMultiPoint(address.getMultiPoint());
rep+   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()];
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.**DBCityObject**

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

```
88    // public JGeometry getPoint(PointProperty pointProperty) {
rep   //    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()) {
rep   //    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()) {
rep   //    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;
rep+  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. From *3D-CityDB* 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 with `PGgeometry.getGeometry` works in similar manner. In *Oracle* `JGeometry` can easily transferred into arrays and processed back again into list-elements for the CityGML-primitives. The ELEM_INFO_ARRAY helps a lot to distinguish between geometric types. The *PostGIS*-JDBC offers different sub-classes from `Geometry.java`. `ComposedGeom` and `MultiLineString` had to be used for addressing subgeometries. Fortunately this didn't lead to conflicts against the names of the `citygml4j.lib`.

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

```
822   // 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**

```
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
rep   // 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
rep   // 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);
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();
```

5c. Synchronization of geometric functions

It's proven that `JGeometry`'s 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 during *PostGIS* processes.

6. How to deal with textures

Pakete:
- api
- cmd
- config
- database
- event
- gui
- log
- modules
- plugin
- util

Klassen:
- [M cityE] DBAppearance
- [M cityE] DBXlinkExporterTextureImage
- [M cityI] XlinkTextureImage

As the data type ORDImage differs a lot from the BYTEA in *PostgreSQL* it's not surprising that the im- and export of textures had to be changed in many aspects. The advantage of ORDImage over common BLOBs is the possibility 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 3d). But all in all the *3D-CityDB* hardly uses the abilities of ORDImage. Even Oracle itself recommended the use of BLOBs for the *3D-CityDB* to the developers.

6a. Import of textures

As seen on the following examples the code for importing textures could be reduced to a few lines. That's because in *Oracle* ORDImages have to be initialized in the database.

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

```
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. Export of textures

Two ways exist for exporting images from the database. No performance-differences could be noticed until now. The first way was preferred as no array with a fixed size had to be declared. This seemed to be more flexible than the second way.

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();
      // }
```

1st way:

```
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;
}
```

2nd 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; }
```

7. The batchsize of PostgreSQL

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

The maximum batchsize of *PostgreSQL* was set to 10000. More might be possible, but was not tested. This change in the Internal-class caused several classes to be changed for compiling. They are all listed in the overview-box.

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;
```

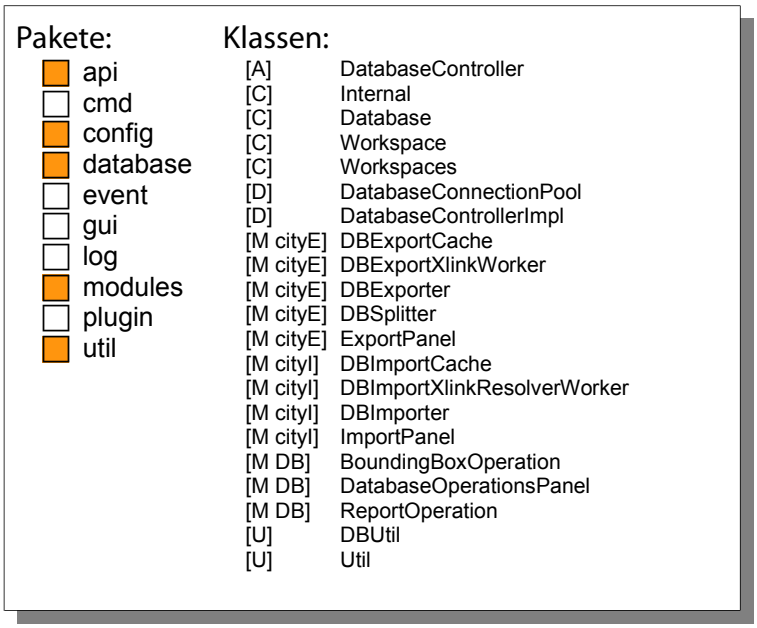
In the following classes no equivalent methods could be found for the Java `PreparedStatement`. The psDrain-batch is now executed and not sent.

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

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

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

8. Workspace Management



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 this in the future. A versioning-approach for *PostGIS* already exists with pgvs. It will be considered for implementation within the next months. Thus the affected packages are colored orange.