3D City Database for CityGML

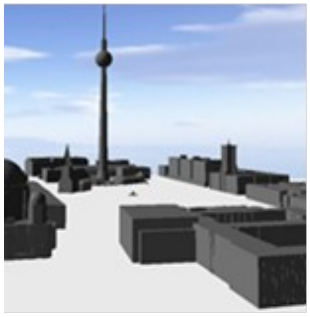
3D City Database Version 2.0.6-postgis

Importer/Exporter Version 1.4.0-postgis

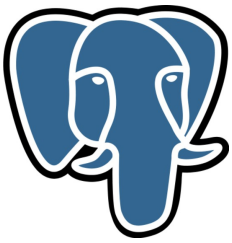
***beta* Version**

Tutorial

16 July 2012







Geoinformation Research Group
Department of Geography
University of Potsdam

Felix Kunde
Hartmut Asche

Institute for Geodesy and
Geoinformation Science
Technische Universität Berlin

Thomas H. Kolbe
Claus Nagel
Javier Herreruela
Gerhard König

(Page intentionally left blank)

Content

Disclaimer:

The *3D City Database version 2.0.6-postgis-beta* and the *Importer/Exporter version 1.4.0-postgis-beta* developed by the *Institute for Geodesy and Geoinformation Science (IGG)* at the *Technische Universität Berlin* is free software under the GNU Lesser General Public License Version 3.0. See the file LICENSE shipped together with the software for more details. For a copy of the GNU Lesser General Public License see the files COPYING and COPYING.LESSER [www1].

THE SOFTWARE IS PROVIDED BY *IGG* "AS IS" AND "WITH ALL FAULTS." *IGG* MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE QUALITY, SAFETY OR SUITABILITY OF THE SOFTWARE, EITHER EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

IGG MAKES NO REPRESENTATIONS OR WARRANTIES AS TO THE TRUTH, ACCURACY OR COMPLETENESS OF ANY STATEMENTS, INFORMATION OR MATERIALS CONCERNING THE SOFTWARE THAT IS CONTAINED ON AND WITHIN ANY OF THE WEBSITES OWNED AND OPERATED BY *IGG*.

IN NO EVENT WILL *IGG* BE LIABLE FOR ANY INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES HOWEVER THEY MAY ARISE AND EVEN IF *IGG* HAVE BEEN PREVIOUSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1. Overview

Welcome to the release of the *3D City Database Version 2.0.6-postgis-beta* and the *Importer/Exporter Version 1.4.0-postgis-beta* for *PostGIS*. With the ability to now store and analyze CityGML-documents in *PostGIS*, we are proud to present our free software in a fully OpenSource-context.

Thanks to the continuous development of *PostGIS* 2.0 with new features for topology, raster and 3D support, a long considered port became feasible. Except for version- and history management all key features of the *3D City Database* incl. the *Importer/Exporter* have been translated to *PostgreSQL / PostGIS.* For a quick overview see table 1.

Please note that this document only gives a short introduction on the *PostGIS*-specific details. For a full overview of the *3DCityDB* and the *Importer/Exporter*, please refer to the version 2.0.1 documentation [1] and the addendum [2] for the recent release of the database and the *Importer/Exporter* tool (2.0.6 and 1.4.0).

Tab. 1: Port-overview on supported key-features of both versions

Key Features of the 3D City Database	**Oracle**	**PgSQL**
Semantically rich, hierarchically structured model	✓	✓
Five different Levels of Detail (LODs)	✓	✓
Appearance data in addition to flexible 3D geometries	✓	✓
Representation of generic and prototypical 3D objects	✓	✓
Free, also recursive aggregation of geo objects	✓	✓
Complex digital terrain models (DTMs)	✓	✓
Management of large aerial photographs	✓	✓
Version and history management	✓	**X**
Matching/merging of building features	✓	✓
Key Features of the Importer/Exporter		
Full support for CityGML 1.0 and 0.4.0	✓	✓
Exports of KML/COLLADA models	✓	✓
Generic KML information balloons	✓	✓
Reading/writing CityGML instance documents of arbitrary file size	✓	✓
Multithreaded programming facilitating high-performance CityGML processing	✓	✓
Resolving of forward and backwards XLinks	✓	✓
XML validation of CityGML documents	✓	✓
User-defined Coordinate Reference Systems	✓	✓
Coordinate transformations for CityGML exports	✓	✓
Matching/merging of building features	✓	✓

✓= equivalent support, ✓= Oracle-specific support ✓= PostGIS-specific support **X** = not supported

3D City Database (abbreviated as 3DCityDB in the following) [3]

- **Complex thematic modelling:**
  Description of thematic features by attributes, relations, nested aggregation hierarchies (part-whole-relations) between features in a semantic and a geometric manner which is useful for thematic queries, analyses, or simulations

- **Five different Levels of Detail (LODs)**
  Multi-representation of geo objects (including DTMs and aerial photographs) in five different LODs based on geometric precision and thematic refinement

- **Appearance data**
  Appearance of features can be used to represent textures and materials, but also non-visual properties like infrared radiation, noise pollution, etc.

- **Complex digital terrain models (DTMs)**
  DTMs can be represented in four different ways: by regular grids, triangulated irregular networks (TINs), 3D mass points and 3D break lines. For each LOD a complex relief can be aggregated from any number of DTM components of different types. For example, 3D mass points and break lines can be used together to form complex terrain models.

- **Representation of generic and prototypical 3D objects**
  For efficient memory management, frequently occurring objects at different locations of the city-model can be stored once for each LOD as a prototype and be referred to as an implicit geometry, e.g. pieces of street, furniture like lanterns, road signs, benches etc.

- **Free, also recursive aggregation of geo objects**
  Geo objects can be aggregated to a group according to user-defined criteria. Each group represents a geo object itself. Names and additional classifying attributes can be assigned to groups. Groups may contain other groups as members, resulting in aggregation hierarchies of arbitrary depth.

- **Flexible 3D geometries**
  Geometries of 3D objects can be represented through the combination of surfaces and solids as well as any, also recursive, aggregation of these elements.

- **Management of large aerial photographs**
  The database can handle aerial photographs of arbitrary size using the new raster2pgsql raster-loader of *PostGIS*

Importer/Exporter

- Import and export of even very large CityGML instance documents (> 4 GB)
- Export of buildings into the KML/COLLAD format
- Coordinate transformation and tiling for exports
- Multiple filter-operations for im- and exports incl. a graphical select of a bounding box by a map widget
- Management of user-defined Coordinate Reference Systems (SRIDs)
- Matching and merging of redundant building objects in the database
- New functionalities can be incrementally added via Plugins

Further information, software downloads, ready-to-use demos, links to the source code repository, and much more can be found at:
http://opportunity.bv.tu-berlin.de/software/projects/3dcitydb-imp-exp/ [www2]
and soon at **the official website of the 3D City Database** [www3].

The *PostGIS* port was realized within a Master's thesis by Felix Kunde conducted at the University of Potsdam, and was supported by the *3DCityDB* developer team of the IGG at the Technical University of Berlin as well as the company virtualcitySYSTEMS GmbH (Berlin, Germany). A previous translation of SQL scripts was done by Laure Fraysse in cooperation with IGO (Paris, France) which was the starting point for the further development.

2. Major Changes to the Oracle version

Data modelling and relational schema

The data model behind the relational database schema of the *3DCityDB* was kept unchanged. Supporting UML diagrams and their relational mapping can be found in the main *3DCityDB* documentation [1]. Only two *Oracle*-specific attribute types had to be changed. First, for polygonal geometries the spatial data type SDO_GEOMETRY was replaced by ST_GEOMETRY (see [1]: 18), and second, the ORD-IMAGE data type for storing texture images was substituted by a simple BLOB (see [1]: 20).

When referring to the relational schema several differences in data types will always occur when using a different Database Management Systems (DBMS). Their internal structure is mostly following the same purpose, so only the name has to be switched. The following table lists the differences between *Oracle Spatial* and *PostgreSQL / PostGIS*:

Tab. 2: Differences in data types

Oracle Spatial	**PostGIS** (PostgreSQL)	**further explanation**
varchar2	varchar	
number	numeric	integer used for refrential id_columns because of serial
binary_double	double precision	
blob, clob	bytea, text	
	serial (integer)	implicitly creates a sequence named `tablename_columnname_seq`
ORDImage	bytea	postgis raster might be an option
sdo_geometry	(st_)geometry	st_geometry also exists in Oracle
sdo_raster	raster	formerly known as WKT Raster
sdo_georaster	raster	

Creating geometric columns and spatial indexes

PostGIS 2.0 introduces the *PostgreSQL* type modifier definition of geometry-columns inside of CREATE TABLE statements [www4]. It was used in the SQL-scripts instead of the older but still common function AddGeometryColumn. Unlike the explicit definition for `USER_SDO_GEOM_METADATA` in *Oracle Spatial*, both methods implicitly insert a tuple of metadata in the `geometry_columns`-view.

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

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

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

or

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

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

The columns that store 3D-geometries are indexed using an n-dimensional GiST index. It is a common practice to force *PostgreSQL* to get rid of dead rows and update table statistics for the spatial columns after bulk inserts or updates and not wait until the "autovaccum-deamon" of *PostgreSQL* will do that [www5] [4]. This is done by the VACUUM ANALYSE command:

```
VACUUM ANALYZE [table_name] [(column_name)];
```

The SQL Planner will use these statistics to evaluate if a GiST index should be used for spatial queries. A SQL script provided to save the user from having to manually write SQL commands for all affected columns. It is shipped within the folder 3dcitydb/postgis/UTIL.

Raster-data management

The raster-data management is much simpler than in *Oracle Spatial*. In *Oracle* a SDO_GeoRaster object must relate to a raster-data-table (RDT) with SDO_Raster objects which hold the actual raster-files. The *Oracle* version of the *3DCityDB* also contains tables for initial imports of image-tiles which can be merged to one raster-file in a second step (IMP-tables). They are necessary for a raster-import tool that was developed for a former version of the *3DCityDB* [5]. *PostGIS 2.0* offers a simple but powerful tool for importing raster-files into

the database called *raster2pgsql*. It is executed from the command line and creates a proper SQL insert command for the selected raster-file. Operators can and should be used for setting the reference system, tiling and different levels for raster-overviews (pyramid-layers) (see example below). In the *PostGIS* approach every stored tile is a raster-object itself, even the raster-overviews. They can be grouped by their original file-name stored in a separate column. This concept would make the RDT and IMP-tables obsolete. Therefore the RDT- and IMP-tables were dropped for the *PostGIS* version of the *3DCityDB*. We recommend to use the *raster2pgsql* tool.

An import into the `raster_relief`-table could look like this:

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

`-f` sets the name of the target column (in this case `rasterproperty`), the target table is specified at last (`raster_reflief`)

`-s` sets the srid for the raster

`-F` adds a column for the original filename (file.format)

`-t` tiling-operator

`-l` levels for raster-overviews

It is possible to import multiple raster-files from a given folder like in the example (`relief/*.tif`). For further readings please refer to the *PostGIS* documentation on raster data management [www6].

History Management

Based on the *Oracle Workspace Manager* it is possible to manage concurrent versions or planning scenarios of the *3DCityDB* within one user schema. They are organized as views of the original dataset or of their parent version. The Oracle version of the *3DCityDB* delivers scripts to enable or disable versioning support for the database tables as well as a bundle of scripts and tools for managing planning-alternatives called the *Planning Manager.* Unfortunately, as *PostgreSQL* does not offer any equivalent facility, the *Planning Manager* and related scripts could not be ported. Corresponding elements in the graphical user interface (GUI) of the *Importer/Exporter* were removed.

A few free projects exist which implement script-based solutions [www7, www8], but for features like the *Planning Manager* they would need a lot of code-rework to get the same results like with Oracle's *Workspace Manager*. It will be considered for future releases of the *PostGIS* version.

Oracle packages vs. PostgreSQL schemas

The *3DCityDB* provides PL/pgSQL stored procedures which are used by the *Importer/Exporter*-tool. Fortunately *PostgreSQL*'s procedural language of SQL PL/pgSQL comes close to Oracle's PL/SQL grammar which facilitated the porting of scripts. Note that previous self-developed scripts for the *Oracle* version will not with *3DCityDB v2.0.6-postgis*. They have to be translated to PL/pgSQL first in order to work correctly.

For the *Oracle* version the procedures and functions were grouped into packages. However, regarding *PostgreSQL* the package concept only exists in the commercial *Plus Advance Server* by EnterpriseDB. An alternative grouping mechanism for stored procedures that is suggested by the *PostgreSQL* documentation [www9] and which has been implemented, is the usage of schemas. A schema is a separate namespace with own tables, views, sequences, functions etc. The packages from the *Oracle* release are represented in one *PostgreSQL* schema called `geodb_pkg` and not in several schemas for each package (see also figure 2 on page 16). But for a better overview the functions were given name prefixes:

Tab. 3: Function-grouping in Oracle and PostgreSQL

former package name	Prefix	Count	Source (PL_pgSQL/GEODB_PKG/)
geodb_delete_by_linage	del_by_lin_	1	DELETE/DELETE_BY_LINAGE.sql
geodb_delete	del_	48	DELETE/DELETE.sql
geodb_idx	idx_	16	INDEX/INDEX.sql
geodb_match	match_	12	MATCHING/MATCH.sql
geodb_merge	merge_	9	MATCHING/MERGE.sql
geodb_stat	stat_	1	STATISTICS/STAT.sql
geodb_util	util_	12	UTIL/UTIL.sql

3. Requirements

This chapter provides an overview of the minimum requirements for the *3DCityDB* and the *Importer/Exporter* tool. Please carefully review these requirements.

3D City Database

As illustrated in chapter 2 some of the features of *PostGIS 2.0* are used. Thus the SQL-scripts would only work with version *2.0* or higher. *PostGIS 2.0* requires *PostgreSQL 8.4* or higher. For 64-bit Windows OS only *9.0* or higher can be used. An empty *3DCityDB* requires 14 MB of disk-space (11 MB *PostGIS* + 3 MB *3DCityDB*).

Importer/Exporter

The *Importer/Exporter* tool can run on any platform providing support for Java 6. It has been successfully tested on (but is not limited to) the following operating systems: Microsoft Windows XP, Vista, 7; Apple Mac OS X 10.6; Ubuntu Linux 9, 10, 11.

Prior to the setup of the *Importer/Exporter* tool, the Java 6 Runtime Environment (JRE version 1.6.0_05 or higher) or Java 7 Runtime Environment (JRE version 1.7.0_03 or higher) must be installed on your system. The necessary installation package can be obtained from [www10].

The *Importer/Exporter* tool is shipped with a universal installer that will guide you through the steps of the setup process. A full installation of the *Importer/Exporter* including documentation and example CityGML files requires approx. 110 MB of hard disk space. Installing only the mandatory application files will use approx. 16 MB of hard disk space. Installation packages can be chosen during the setup process.

The *Importer/Exporter* requires at least 256 MB of main memory. For the import and export of large CityGML respectively KML/COLLADA files, a minimum of 1 GB of main memory is recommended.

4. How to set up a 3DCityDB in PostGIS

Follow these steps to successfully set up a *3DCityDB* on your database server:

1. Installed RDBMS and configuration

Make sure that the *PostgreSQL*-Server-Installation is of version 8.4 or higher. For the right settings of the configuration files check the according *PostgreSQL*-Online-Documentation [www11]. The *PostGIS*-Extension must be of version 2.0.0 or higher. It has to be considered that both projects are under continuous development, but it is recommended that only officially released installers should be used.

2. Run the 3DCityDB-Importer-Exporter-1.4-postgis-Setup

The installer-setup of the software is mostly self-explaining. The SQL and PL/pgSQL scripts of the *3DCityDB* are grouped in the 3dcitydb folder at the target installation-path. The folder structure is explained shortly with the next table:

Tab. 4: Folder hierarchy of the *3DCityDB* installation package

3dcitydb/postgis	**Explanation**
CREATE_DB.bat	batchfile that calls CREATE_DB.sql
DROP_DB.bat	batchfile that calls DROP_DB.sql
CREATE_DB.sql	calls SQL-scripts for setting up the relational schema of the 3DCityDB
CREATE_GEODB_PKG.sql	creates a separate schema in the database named geodb_pkg with stored procedures, called by CREATE_DB.sql
DROP_DB.sql	drops all the schema-elements cascadingly, called by DROB_DB.bat
READ_BEFORE_CREATE.txt	a brief instruction
SCHEMA	called by CREATE_DB.sql
CONSTRAINTS	contains a file that sets the referential foreign keys between tables
INDEXES	contains files for setting sequential (B-Tree) and spatial indexes (GiST)
TABLES	contains files for creating the database tables
PL_pgSQL/GEODB_PKG	called by CREATE_GEODB_PKG.sql
DELETE	contains scripts that help to delete single features from the database. Used by the Matching-Merging-Plugin for the Importer/Exporter.
INDEX	contains scripts with index-functions. Only used by the Importer/Exporter.
MATCHING	contains scripts for the Matching-Merging-Plugin.
STATISTICS	contains a script that generates a database-report for Importer/Exporter
UTIL	contains several helper-functions, mostly for Importer/Exporter
UTIL	called by CREATE_DB.sql
CREATE_DB	contains a script that inserts rows in the objectclass-table
RO_USER	contains a script that creates a read-only user for the database
VACUUM	contains a script that collects table statistics (vacuum) for spatial columns

3. CREATE an empty PostGIS-database

Select a user with privileges to create an empty database with *PostGIS* Extension and also access the *PostGIS* features. No violation of rights should occur when working as a superuser. In the end it should look like in figure 1. The *3DCityDB* will be stored in the public schema, which also contains the *PostGIS* elements like functions, view for spatial metadata and a table for reference systems.

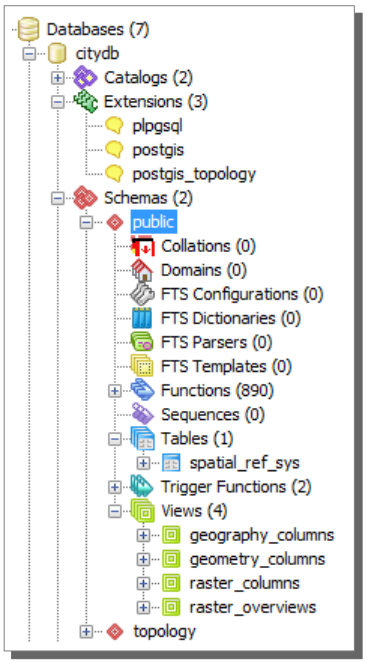


Fig. 1: Empty PostGIS 2.0-database in the pgAdminIII-tool

4. Set up a 3DCityDB

Afterwards, a blank *PostGIS* database is ready to be set up with the relational schema of the *3DCityDB*. The CREATE_DB-script in the main folder 3dcitydb/postgis has to be executed from the *psql* console of *PostgreSQL*. It can not be done in the *pgAdminIII* tool. A more comfortable way is offered with the CREATE_DB batchfile that calls the corresponding SQL file. But it has to be edited first before it is executable. The same applies to the DROP_DB.bat.

```
set PGPORT=5432                 (this is the default port)
set PGHOST=your_host_address    (localhost or server-address)
set PGUSER=your_username        (username e.g. postgres)
set THEDB=your_database_here    (name of the 3D-CityDB e.g. citydb)
set PGBIN=path_to_psql.exe      (e.g. 'PostgreSQL\bin' or 'pgAdmin III')

REM creating 3DCityDB database
"%PGBIN%\psql" -d "%THEDB%" -f "CREATE_DB.sql"

pause
```

Before any script is called the user is asked to define the reference system for the database. Two mandatory parameters have to be set: the SRID and the GML identifier. **If no numeric-value was entered for the SRID the creation of the *3DCityDB* will not run correctly!** No geometric columns will be added to the tables and thus no spatial indexes are possible. If a wrong reference system was chosen when creating the *3DCityDB* it can still be changed with the function `util_change_db_srid(srid NUMERIC, gml_ident VARCHAR)` in the GEODB_PKG schema. Of course, the database should still be empty when the SRID is changed to avoid any false projections or errors on your data. It is also possible to reuse an empty *3DCityDB* as a template for any CityGML model. In case that the SRID is different, just apply the util_change_db_srid script to the new database.

A test session can look like this:

```
path_to_your_importer_exporter_installation\resources\3dcitydb\postgis>
"C:\PostgreSQL\bin\psql" -d "citydb" -f "CREATE_DB.sql"
Password:
SET
Please enter a valid SRID (e.g., 3068 for DHDN/Soldner Berlin): 3068
Please enter the corresponding SRSName to be used in GML exports (e.g.
urn:ogc:def:crs,crs:EPSG::3068,crs:EPSG::5783):
urn:ogc:def:crs,crs:EPSG:6.12:3068,crs:EPSG:6.12:5783
CREATE TABLE
ALTER TABLE
INSERT 0 1
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
...
CREATE INDEX
...
CREATE SCHEMA
CREATE FUNCTION
...
CREATE FUNCION
path_to_your_importer_exporter_installation\resources\3dcitydb\postgis>pause
Press any key to continue . . .
```

After running the CREATE_DB_script the the *3DCityDB* should look like in figure 2. By the counters you can check if the setup was correct.

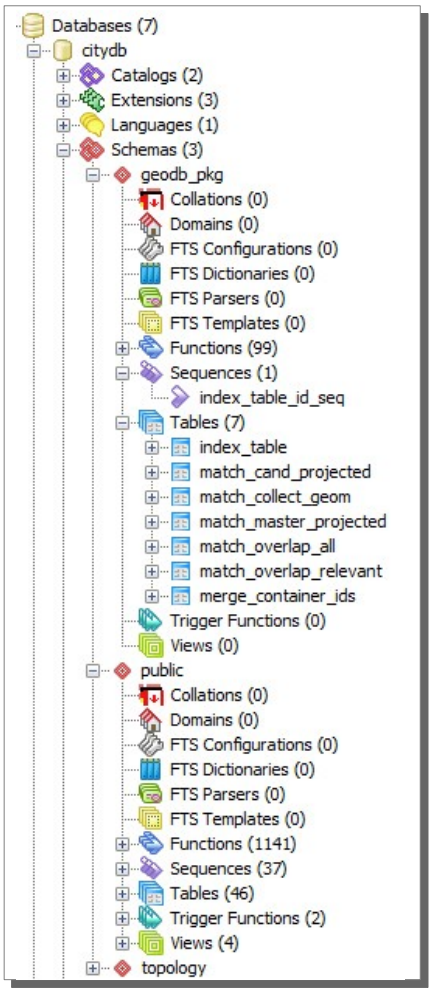


Fig. 2: 3DCityDB in the pgAdminIII-tool

The function util_change_db_srid is executed like this:

```
SELECT geodb_pkg.util_change_db_srid(4326,'urn:ogc:def:crs:EPSG:4326');
```

5. Start the Importer/Exporter

Now that the *3DCityDB* is ready to use, start the *Importer/Exporter* batchfile. After a few seconds the GUI should pop up. Switch to the database-panel and enter your connection details. If you have worked with the *Importer/Exporter* before you will notice that the functionalities are similar to in the *Oracle* version. One difference on the connection details appears at the textfield for the database-name. For connecting to *Oracle* the instance SID had to be entered. For *PostgreSQL / PostGIS* use the name of the database you have created in step 3. If the connection could be established the console-window should look like this:

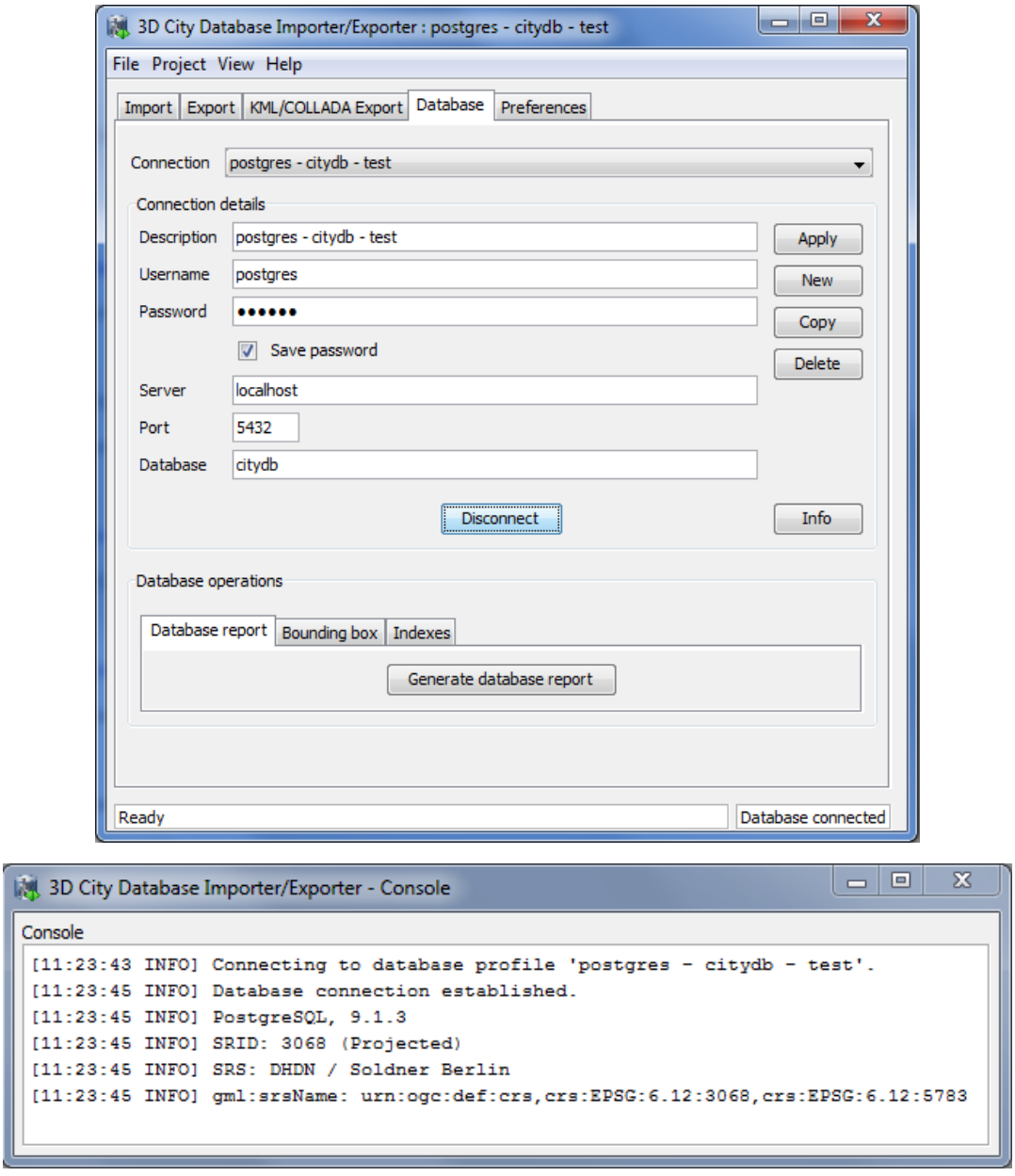


Fig. 3: Importer/Exporter successfully connected to the database (console-window detached)

Again: For further instructions on how to work with the *Importer/Exporter* please read the official *3DCityDB* documentation [1] and the recent addendum [2].

6. DROP the 3DCityDB

When executing the edited DROB_DB.bat all parts of the *3DCityDB* will be deleted from your *PostGIS* database. Note that the database itself remains on the *PostgreSQL* server. It can be reused for another *3DCityDB* of the same version (or higher). However future releases will offer scripts to perform a *3DCityDB* schema update on your database.

5. FAQ

I can not get a connection from the Importer/Exporter to the database. Any help?

Check if you have misspelled some parameter of the connection details. If you are working in a network, check if the *PostgreSQL* configuration file hba.conf contains the address of your client.

Which version of CityGML is supported?

The relational database schema is derived from CityGML version 1.0.0 [6] and is also backwards-compatible with the OGC Best Practices version 0.4.0. The recent release of CityGML 2.0.0 [7] and any Application Domain Extension (ADE) are not yet supported.

How good or bad is the performance of the PostGIS version compared to the Oracle version?

Fair enough. Several im- and exports of CityGML documents of various sizes and contents were tested with both versions. Different city-models were exported to KML/COLLADA, too. In most cases the execution times for the *PostGIS* version reached the same level like the *Oracle* version even with default settings for *PostgreSQL*. It could be noted that untextured CityGML exports were much quicker in *Oracle*. For very large datasets (> 10 GB) *PostgreSQL / PostGIS* scales better for CityGML im- and exports. A detailed analysis comparing the performance of both version is part of the Master's thesis by Felix Kunde [8]. A pdf (german and english) will be available in fall 2012 when the thesis is finished.

Is there a detailed documentation how the port to PostGIS was realized?

A detailed documentation for porting PL/SQL scripts and *Oracle*-specific parts of the Java-code is also shipped with this release. These documents may also provide an introduction for porting own developed features or functions. The Master's thesis will also discuss all aspects on the *PostGIS*-Port in detail.

Which external tools can I use for visualizing my database?

With the *KML-Exporter* the *Importer/Exporter* took advantage of the widespread *Google Earth* client. You are able to export and watch footprints, extruded footprints, geometries and COLLADA files of your buildings on the virtual globe. You can also switch on a mouse-over highlighting and information ballons for the buildings in the preferences for the *KML-Exporter*.

For CityGML you can use several free viewers e.g. *LandXplorer.* If you want to visualize your data directly from the database you can use the *FME-inspector* in connection with *FME PostGIS-reader*. The next version of the OpenSource-GIS *gvSIG* (1.12) will be able to load 3D-Models from a *PostGIS* database.

It happens, that external programs with *PostGIS* drivers try to read the geometries with the deprecated *PostGIS* function `asewkb`, which is now called `ST_AsEwkb`. A bundle of lately deprecated functions can be loaded back into your database by executing the legacy.sql file found in the folder PostgreSQL/share/contrib/postgis-2.0. It can be expected that this mismatch will not appear in recent software-releases

After the KML-Export buildings are flying above the ground. What is going wrong?

Use the default settings in the preferences for the *KML-Exporter* for altitude and terrain. The *KML-Exporter* fetches the heights of Google's elevation-model to calculate the right offset to the buildings in the database. This is also written to the console-window. It is only done once for each building, as the offset is inserted as an generic attribute for the city objects. If you are using CityGML instance documents which were formerly stored in a *Oracle 10g* database and used for KML exports, these entries are holding heights that will not fit *Oracle 11g* or *PostGIS* database. The coordinate transformation to WGS 84 leads to different height-results between *Oracle 10g* and *11g*. *PostGIS*' `ST_Transform` calculates the same values than *Oracle 11g*. To sum it up: Delete the affected rows in the table `Cityobject_GenericAttrib` (with attrname 'GE_LoD[1,2,3 or 4]_zOffset') and restart the KML-export. If facing the message OVER_QUERY_LIMIT the Limit (2500) for requesting heights from Google's elevation service was exceeded and no values will be written in the database. The user has to wait 24 hours to be able to send new requests to the web service with the same client.

I think I've found a bug ...

If so, please report this bug to us. If you have any further issues on the software performance, results of im- and exports or just questions please tell us. We're glad to receive any feedback. Please note that this release is still a beta version. Even though it was tested thoroughly with various datasets of different size and content we cannot guarantee that no errors occur during imports and exports. Please consider testing the PostGIS port in a dedicated testing environment first.

6. References

Documents:

[1] KOLBE, T.H. ; KÖNIG, G. ; NAGEL, C. ; STADLER, A. (2009): 3D-Geo-Database for CityGML. Version 2.0.1. Documentation. Berlin.
Accessible under: http://www.3dcitydb.net/index.php?id=1897

[2] KOLBE, T.H. ; NAGEL, C. ; HERRERUELA, J. (2012): 3D-Geo-Database for CityGML. Addendum to the 3D City Database Documentation Version 2.0. Berlin.
Accessible under: http://www.3dcitydb.net/index.php?id=1897

[3] STADLER, A. ; NAGEL, C. ; KÖNIG, G. ; KOLBE, T.H. (2009): Making interoperability persistent: A 3D geo database based on CityGML. In: LEE, J. ; ZLATANOVA, S. (Ed.): 3D Geoinformation Sciences. Lecture Notes in Geoinformation and Cartography. Springer, Berlin / Heidelberg. 175-192.

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

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

[6] GRÖGER, G. ; KOLBE, T.H. ; CZERWINSKI, A. ; NAGEL, C. (2008): OpenGIS City Geography Markup Language (CityGML) Encoding Standard. Version 1.0.0. OGC 08-007rl.
Accessible under: http://www.opengeospatial.org/standards/citygml

[7] GRÖGER, G. ; KOLBE, T.H. ; NAGEL, C. ; HÄFELE, K-H. (2012): OGC City Geography Markup Language (CityGML) Encoding Standard. Version 2.0.0. OGC 12-019.
Accessible under: http://www.opengeospatial.org/standards/citygml

[8] KUNDE, F. (2012): CityGML in PostGIS – Port, usage and performance-analysis using the example of the 3DCityDB of Berlin. Master Thesis. Not yet finished.

Links:

www1 http://www.gnu.org/licenses/
www2 http://opportunity.bv.tu-berlin.de/software/projects/3dcitydb-imp-exp
www3 http://www.3dcitydb.net
www4 http://postgis.refractions.net/docs/AddGeometryColumn.html
www5 http://postgis.refractions.net/documentation/manualsvn/using_postgis_dbmanagement.html#gist_indexes
www6 http://www.postgis.org/documentation/manual-svn/using_raster.xml.html
www7 http://www.kappasys.ch/cms/index.php?id=23
www8 http://pgfoundry.org/projects/temporal/
www9 http://www.postgresql.org/docs/9.1/interactive/plpgsql-porting.html
www10 http://www.java.com/de/download
www11 http://www.postgresql.org/docs/

Appendix:

List of tables and figures: