# 3D City Database Utilities Package

## PostgreSQL Version

## Documentation

Last update: 28 August 2017

## Active participants in development

| Name | Institution | Email |
|---|---|---|
| Giorgio Agugiaro | AIT – Austrian Institute of Technology G.m.b.H.<br>Center for Energy – Smart Cities & Regions | giorgio.agugiaro@ait.ac.at |

## Acknowledgements

The 3D City Database Utilities Package (PostgreSQL version) was partially developed in the framework of the following international project:

The JPI Urban Europe, ERANET Co-fund Smart Cities/Urban Futures 2015 "**IntegrCiTy**" project (FFG Project number 855078), funded (in Austria) by the Austrian Ministry for Transport, Innovation and Technology and with the support from the European Union's Horizon 2020 research and innovation programme.

Homepage: http://integrcity.epfl.ch

# Table of Contents

## Disclaimer

The *3D City Database Utilities Package,* developed by Austrian Institute of Technology, Center for Energy, Smart Cities and Regions Research Field (AIT), is free software and licensed under the Apache License, Version 2.0. See the file LICENSE file shipped together with the software for more details. You may obtain a copy of the license at http://www.apache.org/licenses/LICENSE-2.0.

THE SOFTWARE IS PROVIDED BY *AIT* "AS IS" AND "WITH ALL FAULTS." *AIT* 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.

*AIT* 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 *AIT*.

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

# Introduction

The 3D City Database Utilities Package contains a number of additional stored procedures, views and trigger functions which extend the “vanilla” version of the 3DCityDB. The Utilities Package is intended to be installed optionally on top of a 3DCityDB database instance, and the overall goal is to offer additional features and functionalities which help in the data management tasks.

As a matter of fact, data handling within the 3DCityDB can be sometimes not immediately straightforward due to the complexity of the database schema (which indeed reflects the richness and complexity of the CityGML data model). For example, data belonging to a specific class can be stored in different linked tables in the database. Therefore CRUD[1] operations (i.e. select, insert, update and delete) may result in complex SQL statements. In order to partially overcome this complexity, the 3DCityDB is already shipped with a number of stored procedures, e.g. the “delete” ones. These are of particular help as the complexity of the ordered, hierarchical deletion process is carried out automatically and hidden from the user.

[1] CRUD represents an acronym for the database operations **C**reate, **R**ead, **U**pdate, and **D**elete.

# 1 Content

The Utility Package for the 3D City Database provides in the current release:

- Insert stored procedures for some of the “vanilla” 3DCityDB tables;
- Views offering a simplified access to data spread over multiple tables. All views are created in a separated database schema named *citydb_view*;
- Additional “smart” CRUD stored procedures (for insert and delete operations) in the *citydb_view* schema;
- Trigger functions built upon the aforementioned “smart” stored procedures which make most of the views updatable. As a consequence, the user can interact directly with the views as if they were normal tables and perform not only select operations, but also insert, update and delete.

New and improved functionalities of the Utilities Package will be added in future releases as developments continues.

## 1.1 Insert stored procedures in schema citydb_pkg

The insert stored procedures allow to insert data into the table they refer to. They are stored in the *citydb_pkg* schema and are named using the table name prefixed with “insert_”, e.g. `citydb_pkg.insert_building(…)` or `citydb_pkg.insert_cityobject(…)`.

All stored procedures (also called “functions”, in PostgreSQL) are written in PL/pgSQL and they all have parameters that are called using the named notation[2]. The named notation is especially useful for functions that have a large number of parameters, since it makes the associations between parameters and actual arguments more explicit and reliable. In the named notation, the arguments are matched to the function parameters by name and can be written in any order. Parameters that have default values given in the function declaration need not to be written at all in the call. This is particularly useful since any combination of parameters can be omitted. In order to use an insert function with the named notation, each argument's name is specified using **:=** to separate it from the argument expression. For example:

```sql
SELECT citydb_pkg.insert_building(
id := 5094,
building_root_id := 5094,
class := ‘Residential’,
storeys_above_ground := 5,
storeys_below_ground := 2
);
```

The insert stored procedures have the following characteristics:

[2] The other approach is called *positional notation*. For more details, please refer to https://www.postgresql.org/docs/9.1/static/sql-syntax-calling-funcs.html

- All parameters are defined as DEFAULT NULL, i.e. they are all optional, except for:
    - The OBJECTCLASS_ID parameter, if the table has such attribute (e.g. table CITYOBJECT);
    - The ID parameter, if the table has a primary key which is *not* generated by a sequence (e.g. tables BUILDING);
    - Both parameters forming the primary key of an association table (e.g. GROUP_TO_CITYOBJECT). Such association tables can be easily recognised by the "_TO_" string in the table name;
- For tables having the ID generated by a sequence, there are two possibilities:
    - The user does not specify the ID parameter. In this case the next available value is retrieved from the corresponding sequence automatically and assigned to the new record to be inserted;
    - The user specifies the ID parameter. In this case, it is the responsibility of the user to choose the ID value properly, in order to comply with the UNIQUE constraint in the corresponding table. If the ID value is already used, an error will be raised and the insert operation aborted.
- If the GMLID parameter is not given by the user, then a universally unique identifier (UUID) is generated automatically and assigned to the new record to be inserted;
- Upon successful completion, the insert stored procedures return the ID of the inserted record. In case of an association table, the stored procedure returns null.
- In case of an EXCEPTION, a notice is raised with the error message.

## 1.2 Views

As mentioned before, a number of views is installed in the database schema *citydb_view*. Regarding the **naming convention**, most of the views allow to access data for specific object classes. The name of the view coincides or approximates the name of the corresponding class in the OBJECTCLASS table. For example, view BUILDING presents data of Building objects, while view BUILDING_PART presents data of BuildingPart objects. In addition, wherever appropriate, not only the OBJECTCLASS_ID attribute, but also the **CLASSNAME** one (retrieved from the joined OBJECTCLASS table) are included in the view.

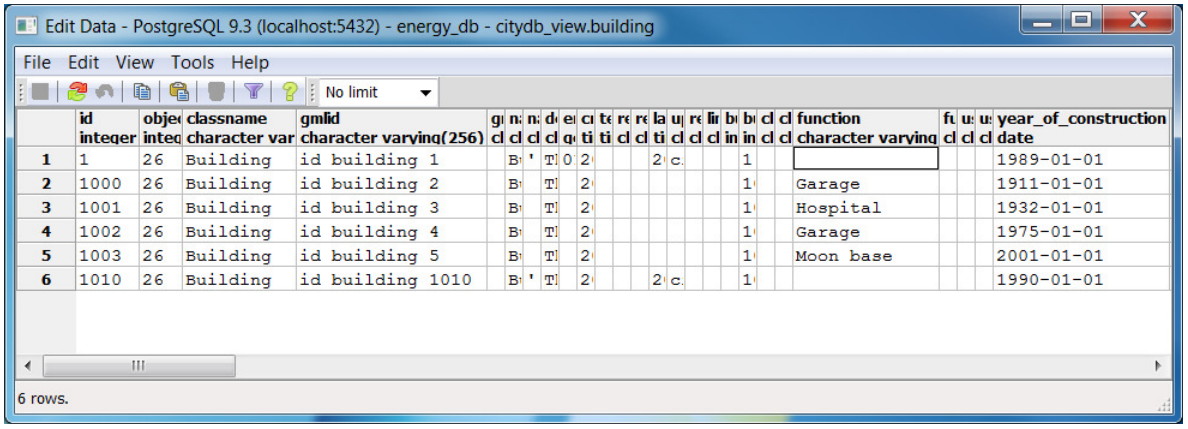
**Please note:** in order to access *any* database object in the *citydb_view* schema, **qualified names**[3] consisting of the schema name and the object name separated by a dot ***must* be used**. This is a design decision, in order to avoid homonymy issues since some objects (e.g. views) have the same names of other objects in the *citydb* and *citydb_pkg* schemas. As a matter of fact, the *search_path* variable of the database instance remains unchanged to the default values (i.e. search_path = citydb, citydb_pkg, public).

Table 1 lists the views shipped with this version of the 3DCityDB Utilities Package, while Figure 1 presents an example of view BUILDING.

[3] For more details, see https://www.postgresql.org/docs/9.1/static/ddl-schemas.html

| VIEW | UPDATABLE? |
|---|---|
| building | ✓ |
| building_furniture | |
| building_installation | |
| building_installation_exterior | |
| building_installation_interior | |
| building_lod0_footprint | |
| building_part | ✓ |
| cityobjectgroup | ✓ |
| generic_cityobject | |
| group_to_cityobject | |
| land_use | |
| opening | |
| opening_door | |
| opening_window | |
| plant_cover | |
| room | |
| solitary_vegetat_object | |
| thematic_surface | |
| thematic_surface_ceiling | |
| thematic_surface_closure | |
| thematic_surface_floor | |
| thematic_surface_ground | |
| thematic_surface_interior_wall | |
| thematic_surface_outer_ceiling | |
| thematic_surface_outer_floor | |
| thematic_surface_roof | |
| thematic_surface_wall | |
| waterbody | |

**Table 1: Views shipped with the current version of the Utilitites Package. All views are stored in schema *citydb_view*.**


Edit Data - PostgreSQL 9.3 (localhost:5432) - energy_db - citydb_view.building

| | id integer | objec integ | classname character var | gmlid character varying(256) | gı cl | nı cl | nı cl | dı cl | eı q | cı ti | te ti | re cl | re cl | la ti | uı cl | re cl | lir cl | bı in | bı in | cl cl | cl cl | function character varying | fu cl | u cl | u cl | year_of_construction date |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 26 | Building | id building 1 | | B | ' | T | 0 | 2 | | | | 2 | c. | | | | 1 | | | | | | | 1989-01-01 |
| 2 | 1000 | 26 | Building | id building 2 | | B | | T | | 2 | | | | | | | | | 1 | | | Garage | | | | 1911-01-01 |
| 3 | 1001 | 26 | Building | id building 3 | | B | | T | | 2 | | | | | | | | | 1 | | | Hospital | | | | 1932-01-01 |
| 4 | 1002 | 26 | Building | id building 4 | | B | | T | | 2 | | | | | | | | | 1 | | | Garage | | | | 1975-01-01 |
| 5 | 1003 | 26 | Building | id building 5 | | B | | T | | 2 | | | | | | | | | 1 | | | Moon base | | | | 2001-01-01 |
| 6 | 1010 | 26 | Building | id building 1010 | | B | ' | T | | 2 | | | | 2 | c. | | | | 1 | | | | | | | 1990-01-01 |

6 rows.

**Figure 1: Example of view BUILDING**

## 1.3 Additional stored procedures in schema citydb_view

Within the schema *citydb_view* a number of stored procedures is installed. They can be grouped in two groups: delete and insert stored procedures.

The **delete stored procedures** work conceptually like the homologous ones in the *citydb_pkg* schema. They simplify the deletion process, especially in case of complex objects stored in multiple tables. Moreover, they are used by the trigger functions (see later).

The “smart” **insert stored procedures** build upon the aforementioned ones in the *citydb_pkg* schema, however with a number of differences, namely:

a) They allow to insert in one statement all attributes of an object which may be spread over multiple tables. For example, the stored procedure `citydb_view.insert_building(…)` takes as input *all* parameters of the underlying tables CITYOBJECT *and* BUILDING, and takes care of inserting the data appropriately. In this example, data of table CITYOBJECT is first inserted, then data of table BUILDING;
b) The user *must not* provide the OBJECTCLASS_ID value as input parameter of the stored procedure. If required, each stored procedures looks up automatically for the proper OBJECTCLASS_ID in table OBJECTCLASS and uses it. For example, `citydb_view.insert_building(…)` automatically retrieves the value 26 and uses it in the following insert statements;
c) For the input of the ID parameters, the approach is the same like in the insert stored procedures in schema *citydb_pkg*: the user can leave it blank (thus automatically getting the next available value from the corresponding sequence), or provide a value. In either case, the ID is used and passed to all tables involved in the cascading insert operation.
d) Upon successful completion, the insert stored procedures return the ID of the inserted record. In case of an association table, the stored procedure returns null.
e) In case of an EXCEPTION, a notice is raised with the error message.

## 1.4 Trigger functions

For nearly all views, a number of triggers and trigger functions are added, in order to make the views updatable. Therefore, if a view is also updatable, entries can be updated, inserted and deleted directly interacting with the view. Whether a view is updatable or not is shown in Table 1. Please note that, by design decision, the underlying update trigger functions do *not* allow to change neither the ID nor the OBJECTCLASS_ID attributes.

Although still limited at the time of writing, the number of database object shipped with the 3DCityDB Utilities Package will increase in the upcoming releases of the Utilities Package. It is nevertheless important to remark that the same concepts will apply for database objects shipped with ADEs (views, stored procedures, triggers, etc.), which will build upon some of the objects shipped with this package.

# 2 Installation

The source code and the documentation the 3DCityDB **Utilities Package** for PostgreSQL can be retrieved here: https://github.com/gioagu/3dcitydb_ade/tree/master/00_utilities_package

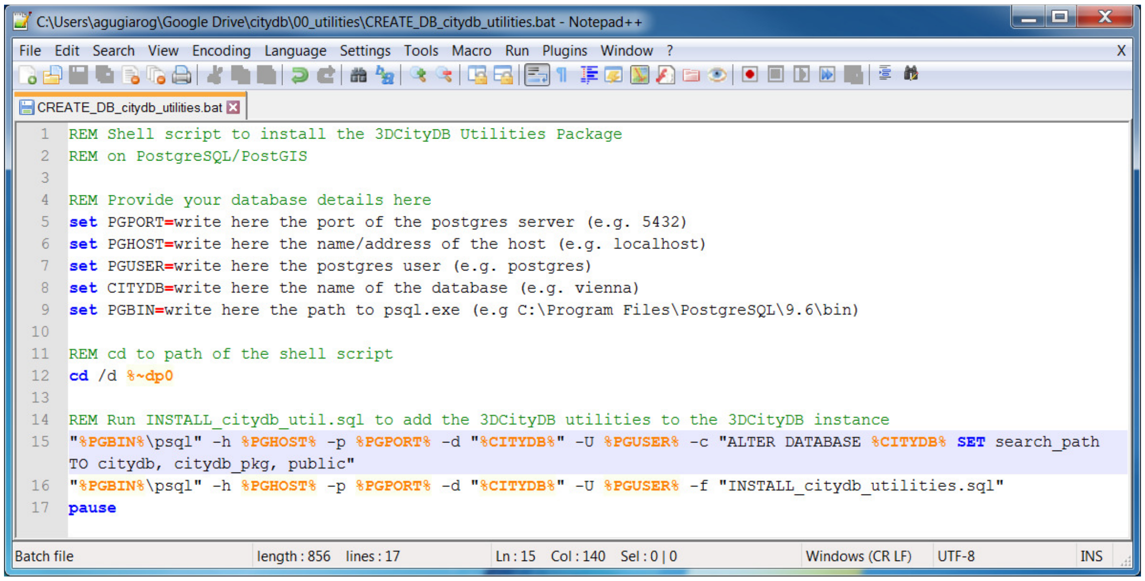
## 2.1 System requirements

In order to set up the 3DCityDB Utilities Package, the 3DCityDB v.3.3.0 needs to have been previously installed correctly. All system requirements of the 3DCityDB still apply. PostgreSQL is supported from v. 9.1. PostGIS 2.0 or higher is required.

## 2.2 Automatic installation

The (recommended) installation process can be carried out nearly completely automatically. The user simply needs to run from a Windows command shell (or simply by double-clicking it) the batch file CREATE_DB_citydb_utilities.bat. However, the configuration parameters regarding the PostgreSQL server and the selected database must be first adapted. A simple text editor will suffice. Please note that the installation should conveniently be carried out using the same user that installed the *citydb* schema.

An example of the CREATE_DB_citydb_utilities.bat file is shown in Figure 2. For Linux environments, the equivalent file CREATE_DB_citydb_utilities.sh is also provided. Upon successful installation, the message shown in Figure 3 will be output.


C:\Users\agugiarog\Google Drive\citydb\00_utilities\CREATE_DB_citydb_utilities.bat - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
CREATE_DB_citydb_utilities.bat
Batch file length : 856 lines : 17 Ln : 15 Col : 140 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

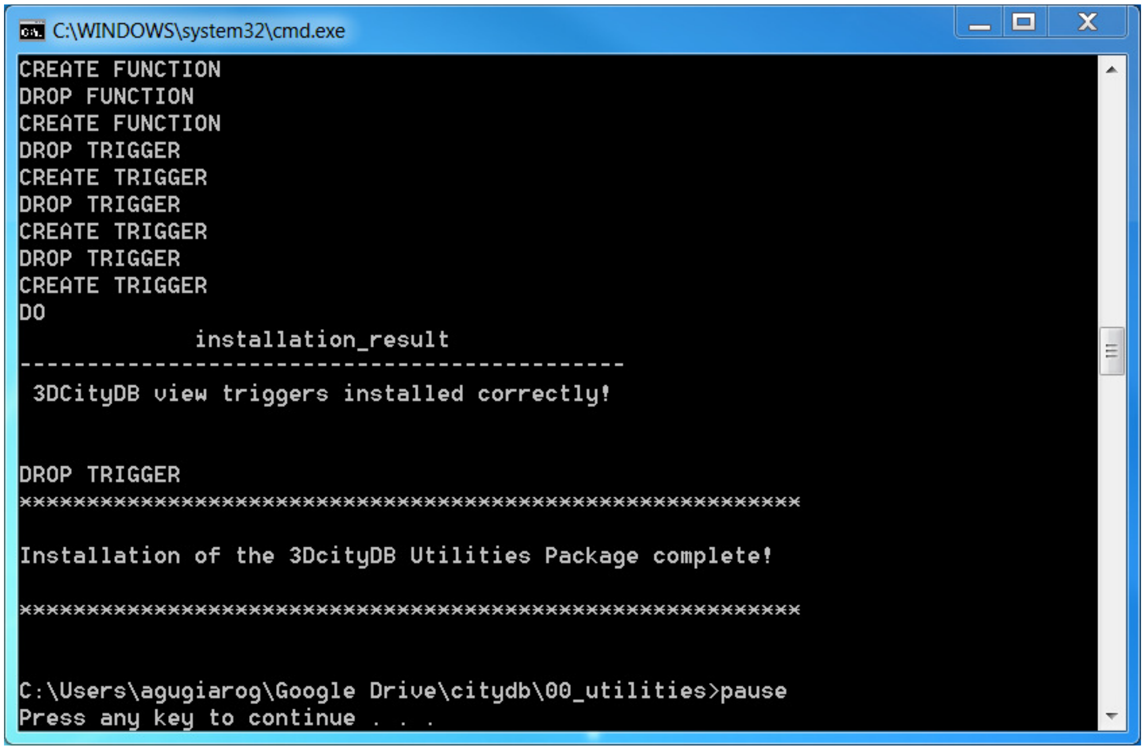
```
REM Shell script to install the 3DCityDB Utilities Package
REM on PostgreSQL/PostGIS

REM Provide your database details here
set PGPORT=write here the port of the postgres server (e.g. 5432)
set PGHOST=write here the name/address of the host (e.g. localhost)
set PGUSER=write here the postgres user (e.g. postgres)
set CITYDB=write here the name of the database (e.g. vienna)
set PGBIN=write here the path to psql.exe (e.g C:\Program Files\PostgreSQL\9.6\bin)

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

REM Run INSTALL_citydb_util.sql to add the 3DCityDB utilities to the 3DCityDB instance
"%PGBIN%\psql" -h %PGHOST% -p %PGPORT% -d "%CITYDB%" -U %PGUSER% -c "ALTER DATABASE %CITYDB% SET search_path
TO citydb, citydb_pkg, public"
"%PGBIN%\psql" -h %PGHOST% -p %PGPORT% -d "%CITYDB%" -U %PGUSER% -f "INSTALL_citydb_utilities.sql"
pause
```

**Figure 2: Example of the CREATE_DB_citydb_utilities.bat batch file**



C:\WINDOWS\system32\cmd.exe

```
CREATE FUNCTION
DROP FUNCTION
CREATE FUNCTION
DROP TRIGGER
CREATE TRIGGER
DROP TRIGGER
CREATE TRIGGER
DROP TRIGGER
CREATE TRIGGER
DO
             installation_result
----------------------------------------------
 3DCityDB view triggers installed correctly!


DROP TRIGGER
***********************************************************

Installation of the 3DcityDB Utilities Package complete!

***********************************************************


C:\Users\agugiarog\Google Drive\citydb\00_utilities>pause
Press any key to continue . . .
```

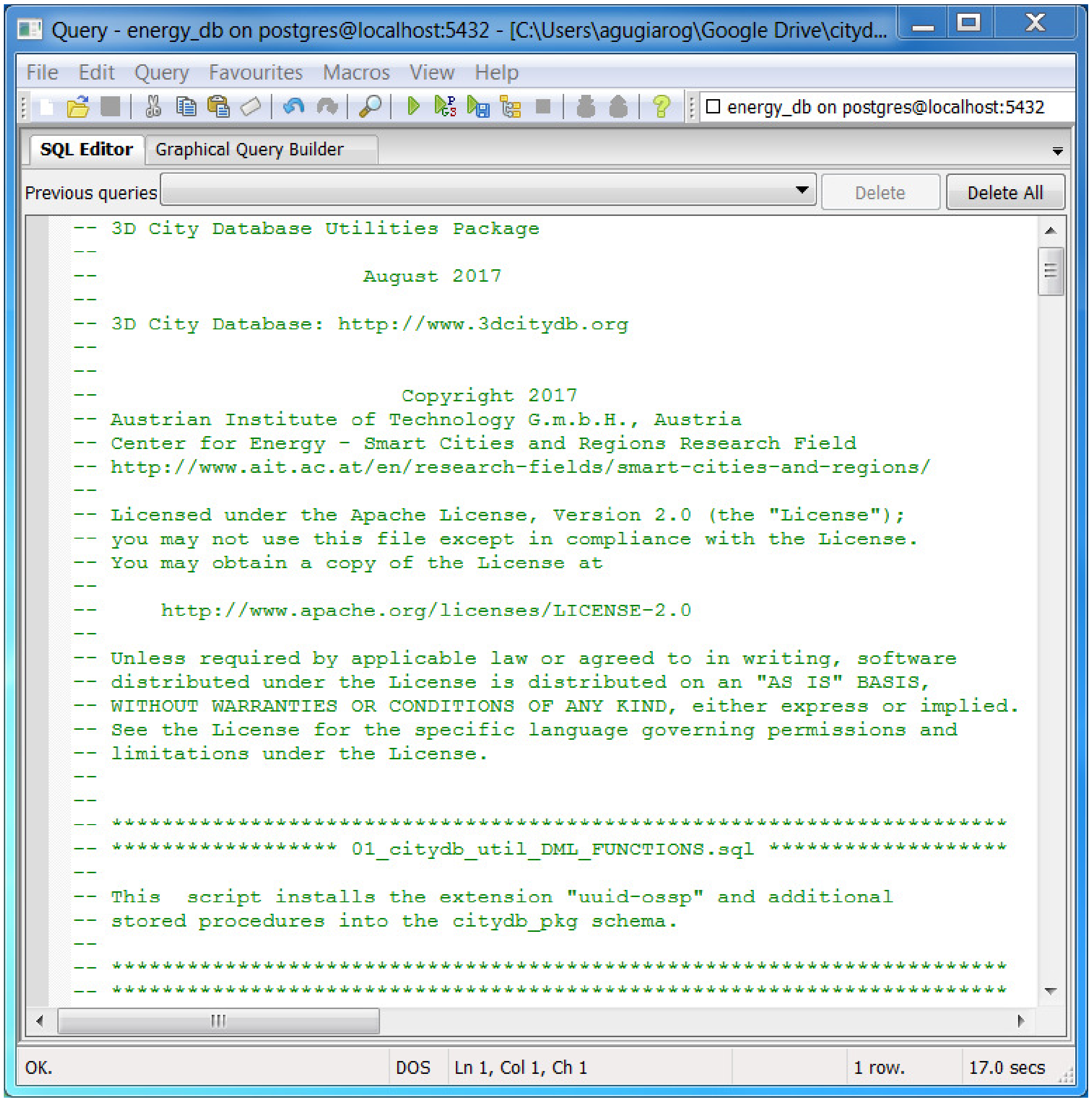
**Figure 3: Message upon successful installation of the 3DCityDB Utilities Package**

## *2.3 (Alternative) manual installation*

As an alternative, manual, installation process, the user must **sequentially** launch the following SQL scripts, using for example the PgAdmin GUI, which is generally installed together with any recent PostgreSQL installation package. See for example Figure 4, where PgAdmin III v. 1.22 is being used. Upon successful installation, messages like the one shown in Figure 5**Error! Reference source not found.** will be output.

The SQL scripts can be found in the \00_utilities\postgresql subfolder and are:

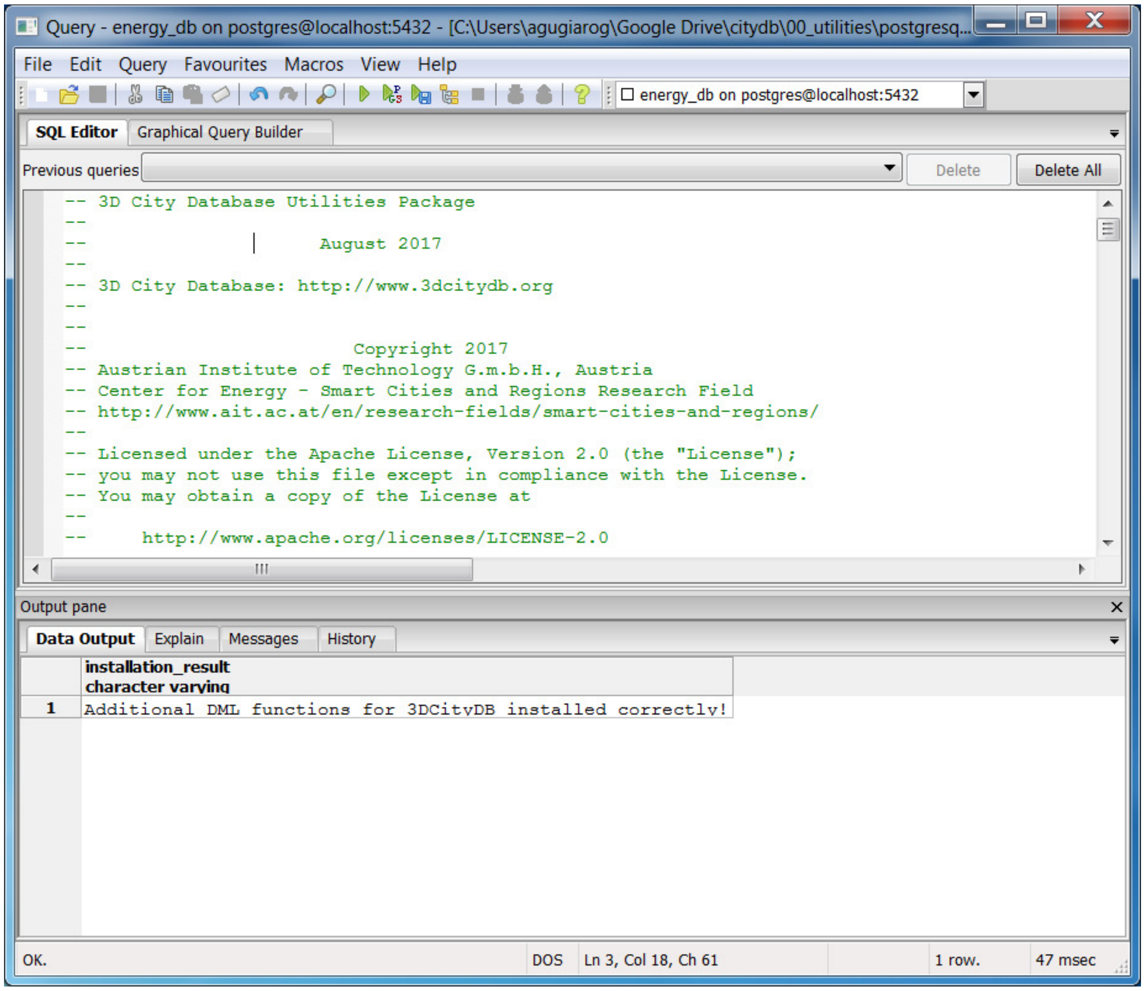
01_citydb_util_DML_FUNCTIONS.sql,
02_citydb_util_VIEWS.sql,
03_citydb_util_VIEW_FUNCTIONS.sql,
04_citydb_util_VIEW_TRIGGERS.sql.


Query - energy_db on postgres@localhost:5432 - [C:\Users\agugiarog\Google Drive\cityd...
File Edit Query Favourites Macros View Help
energy_db on postgres@localhost:5432
SQL Editor
Graphical Query Builder
Previous queries
Delete
Delete All

```
-- 3D City Database Utilities Package
--
--                      August 2017
--
-- 3D City Database: http://www.3dcitydb.org
--
--
--                         Copyright 2017
-- Austrian Institute of Technology G.m.b.H., Austria
-- Center for Energy - Smart Cities and Regions Research Field
-- http://www.ait.ac.at/en/research-fields/smart-cities-and-regions/
--
-- Licensed under the Apache License, Version 2.0 (the "License");
-- you may not use this file except in compliance with the License.
-- You may obtain a copy of the License at
--
--     http://www.apache.org/licenses/LICENSE-2.0
--
-- Unless required by applicable law or agreed to in writing, software
-- distributed under the License is distributed on an "AS IS" BASIS,
-- WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
-- See the License for the specific language governing permissions and
-- limitations under the License.
--
--
-- ***********************************************************************
-- ****************** 01_citydb_util_DML_FUNCTIONS.sql *******************
--
-- This  script installs the extension "uuid-ossp" and additional
-- stored procedures into the citydb_pkg schema.
--
-- ***********************************************************************
-- ***********************************************************************
```

```
OK.    DOS    Ln 1, Col 1, Ch 1    1 row.    17.0 secs
```

**Figure 4: Installation of the Utilities Package using PgAdmin III**


Query - energy_db on postgres@localhost:5432 - [C:\Users\agugiarog\Google Drive\citydb\00_utilities\postgresq...
File Edit Query Favourites Macros View Help
energy_db on postgres@localhost:5432
SQL Editor
Graphical Query Builder
Previous queries
Delete
Delete All
Output pane
Data Output
Explain
Messages
History
OK.
DOS
Ln 3, Col 18, Ch 61
1 row.
47 msec

```
-- 3D City Database Utilities Package
--
--                    August 2017
--
-- 3D City Database: http://www.3dcitydb.org
--
--
--                        Copyright 2017
-- Austrian Institute of Technology G.m.b.H., Austria
-- Center for Energy - Smart Cities and Regions Research Field
-- http://www.ait.ac.at/en/research-fields/smart-cities-and-regions/
--
-- Licensed under the Apache License, Version 2.0 (the "License");
-- you may not use this file except in compliance with the License.
-- You may obtain a copy of the License at
--
--     http://www.apache.org/licenses/LICENSE-2.0
```

```
   installation_result
   character varying
1  Additional DML functions for 3DCityDB installed correctly!
```

**Figure 5: Upon successful installation, the resulting message can be read in the lower part of the window (Output pane)**