PLATEAU by MLIT

PLATEAU Technical Report

3D 都市モデル活用のための技術資料

CityGML に最適化されたデータベース構築手法に関する技術調査レポート

series No. 63

Technical Report on Database Construction Methods Optimized for CityGML

目次

1. 実施概要 1
1.1. 本レポートの目的 1
2. 3DCityDB 調査 2
2.1. 3DCityDB の概要 2
2.2. 3DCityDB の作成 4
2.3. データのインポート 7
2.4. データのエクスポート 9
2.5. Importer/Exporter のプラグイン 16
2.6. データベース拡張検討 17
2.6.1. 検討環境 17
2.6.2. データベース拡張 19
2.6.3. CityGML データのインポート 24
2.7. 3DCityDB と Cesium との連携 26
2.7.1. 3DCityDB-Web-Map-Client の機能 28
2.7.2. 3DCityDB-Web-Map-Client と PostgREST の連携 36
2.7.3. PostgREST を使用した DB 更新方法の確認 43
2.8. i-UR 3.0 対応 49
2.8.1. コード解析 50
2.9. CityGML 3.0 対応 56
3. 3DCityDB サーバーの利用手順書 57
3.1. サーバー環境 57
3.2. クライアント環境 58
3.2.1. Importer/Exporter のインストール方法 58
3.2.2. A5:SQL Mk-2 のインストール方法 66
3.3. DB 接続方法 67
3.3.1. Importer/Exporter ツールでの DB 接続 67
3.3.2. A5:SQL Mk-2 での DB 接続 70
3.4. CityGML データのインポート・エクスポート方法 76
3.4.1. CityGML データのインポート方法 76
3.4.2. CityGML データのエクスポート方法 85
3.5. 3DCityDB-Web-Map-Client の利用方法 90
3.6. A5:SQL Mk-2 によるデータ更新 94
4. 成果と課題 99
5. 用語集 101
6. 参考資料 103

1. 実施概要

1.1. 本レポートの目的

国土交通省都市局では2020年度からProject PLATEAUを開始し、スマートシティの社会実装をはじめとするまちづくりのデジタルトランスフォーメーションを推進するための基盤データとして、3D都市モデルの整備・活用・オープンデータ化事業を進めている。

3D都市モデルは、モデルの幾何形状や地物定義、モデルの用途や構造などの属性情報を保持するため、1つの3D都市モデルのデータセットが保持するデータ量が多い。また、3D都市モデルは地域メッシュ（緯度経度に基づいて地域を網目のように分割した区域）ごとのCityGMLファイルに保存し、管理されているため、データ容量や検索性、更新の負荷といった課題がある。

膨大なデータ容量を持つ3D都市モデルであるが、現在のデータセット管理はPLATEAU CMSと呼ばれるクラウドシステムによって行われている。PLATEAU CMSでは、3D都市モデルのデータセットを地物単位及び都市単位の圧縮ファイル形式で保管し、必要な品質検査や3DTiles変換などをウェブ上で完結する仕組みとして構築されている。

このような管理方法はデータセット自体の管理やデータ変換などには便利であるが、CityGML自体の検索や更新などを動的に行うものではない。このため、今後さらに膨大な3D都市モデルが整備されていくことを見据えると、CityGMLデータ自体をリレーショナルデータベースとして管理し、データの一元化や検索性の向上、多くのユーザーや地理情報システムとのデータ共有などを行うことができるデータベースシステムの構築について検討する必要がある。

3D都市モデルのデータベース化については、ミュンヘン工科大学の地理情報学講座（the Chair of Geoinformatics, Technical University of Munich）、Virtual City Systems社、M.O.S.S.社（M.O.S.S. Computer Grafik Systeme GmbH）の共同開発による「3DCityDB」が先行研究として存在する。

このため、本レポートでは、3D都市モデルのデータベース管理を実現するための基礎研究として、既往技術である3DCityDBの調査と、実際に構築した3DCityDBの課題をまとめることで、今後の調査研究に向けた技術的ナレッジを提供することを目的とする。

2. 3DCityDB 調査

2.1. 3DCityDB の概要

3DCityDB は、標準的な空間リレーショナルデータベース上で仮想 3D 都市モデルを管理するための無料のジオデータベースである（https://www.3dcitydb.org/）。調査時点では、CityGML 2.0 の規格に対応している。

3DCityDB は、デジタルツインとしても知られる仮想 3D 都市モデルのための高性能でスケーラブルなデータストアを提供するために平成 15 年（2003 年）より開発が開始された。開発当初は、ボン大学の地図・地理情報研究所（Institute for Cartography and Geoinformation at University of Bonn）と lat/lon 社が、平成 18 年（2006 年）からは、ベルリン工科大学の測地学・地理情報科学研究所（Institute for Geodesy and Geoinformation Science, Technische Universität Berlin）と 3DGeo 社（後に Autodesk が買収）が共同で開発を担当した。平成 25 年（2013 年）以降は、ミュンヘン工科大学の地理情報学講座（the Chair of Geoinformatics, Technical University of Munich）、Virtual City Systems 社、M.O.S.S.社（M.O.S.S. Computer Grafik Systeme GmbH）が開発を担当している（https://3dcitydb-docs.readthedocs.io/en/latest/overview/history.html）。

3DCityDB は、ベルリン、ポツダム、ハンブルグ、ミュンヘン、フランクフルト、ドレスデン、ロッテルダム、ウィーン、ヘルシンキ、シンガポール、チューリッヒなどの多くの都市で仮想 3D 都市モデルを管理するために利用されている。また、開発を担当している Virtual City Systems 社と M.O.S.S.社は仮想 3D 都市モデルの作成、保守、視覚化、変換、エクスポートを行うための商用製品やサービスの中核として 3DCityDB を使用している。さらに、ドイツの連邦各州のマッピング機関は、3DCityDB を使用して州全体で収集された仮想 3D 都市モデルの保存と管理を行っている。

3DCityDB の利用にあたりデータベースが必要となる。どのデータベースを使用するかはユーザーの任意となる。

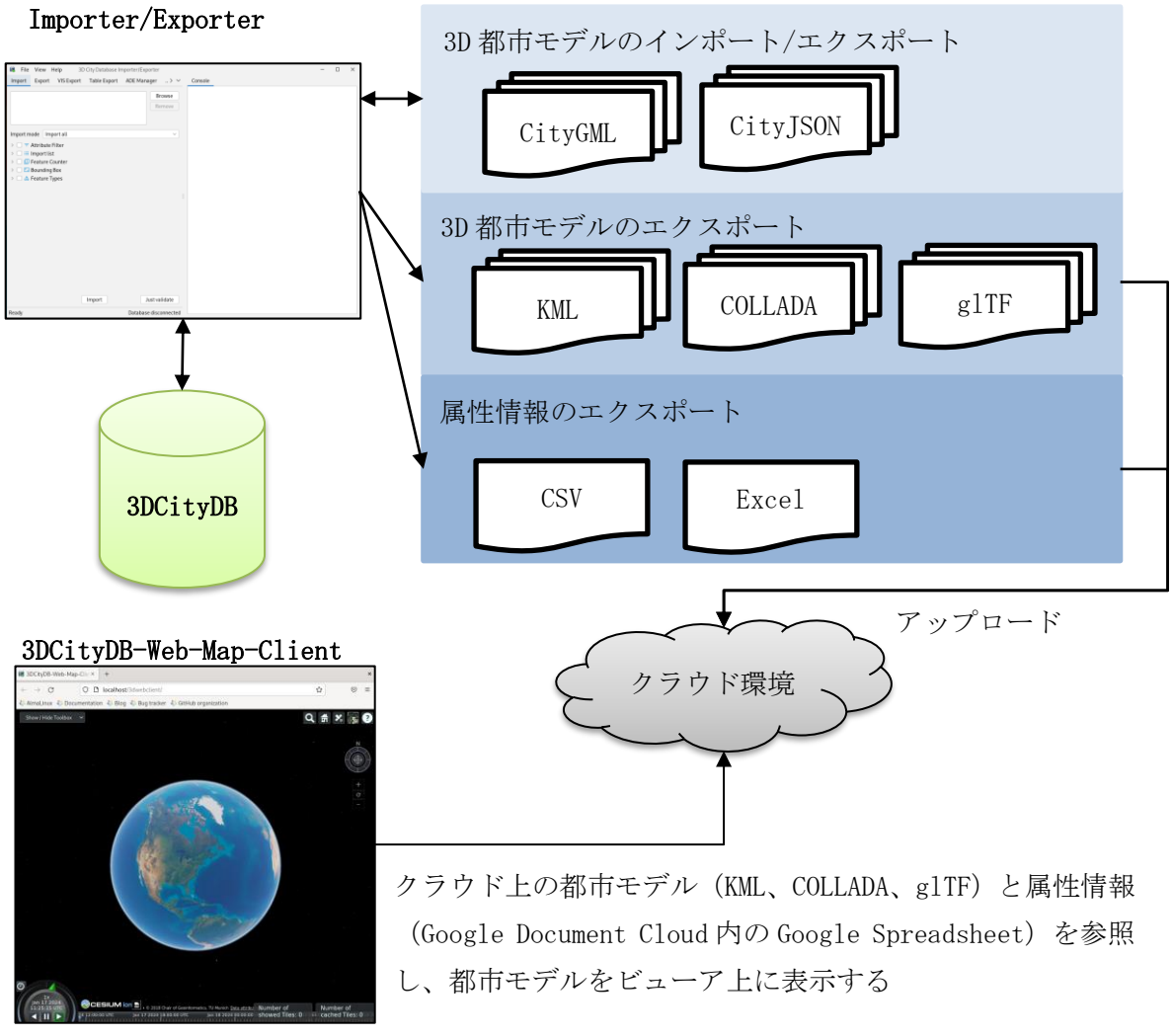
表 2-1 3DCityDB がサポートするデータベース

データベース	バージョン	有償/無償
PostgreSQL（PostGIS 拡張機能付き）	PostgreSQL ver.11 以上、PostGIS ver.2.5 以上	無償
Oracle（Spatial 機能付き）	Oracle ver.19c 以上	有償
PostgreSQL 用の PolarDB（Ganos 拡張機能付き）	PolarDB ver.1.1 以上、Ganos ver.4.6 以上	有償

3DCityDB は、CityGML に対応したデータベースのスキーマが用意されているだけではなく、データ交換やクラウドサービスとの連携を容易にする Importer/Exporter ツール（https://github.com/3dcitydb/importer-exporter）や、都市モデルの可視化用に 3D ウェブビューアアプリである 3DCityDB-Web-Map-Client（https://github.com/3dcitydb/3dcitydb-web-map）が付属している。

3DCityDB と Importer/Exporter を使用すると、CityGML 2.0、及び CityGML 1.0 に準拠した CityGML 又は CityJSON をインポートしデータベース上で 3D 都市モデルを管理することが可能となる。また、Importer/Exporter はデータベース上の 3D 都市モデルをエクスポートする機能を有する。エクスポート機能は 3 種類あり、CityGML 又は CityJSON へのエクスポート、Google Earth、ArcGIS、Cesium などで利用可能な KML、COLLADA、glTF へのエクスポート、属性情報の CSV 又は Microsoft Excel ファイルへのエクスポートが可能である。

3DCityDB-Web-Map-Client は、3D 都市モデルを表示するためのウェブビューアであり、クラウド上に保存された KML、glTF（Impoter/Exporter のエクスポートファイル）を参照して、3D 都市モデルをビューア上に表示することができる（https://3dcitydb-docs.readthedocs.io/en/latest/overview/introduction.html、https://3dcitydb-docs.readthedocs.io/en/latest/overview/main-features.html）。



クラウド上の都市モデル（KML、COLLADA、glTF）と属性情報（Google Document Cloud 内の Google Spreadsheet）を参照し、都市モデルをビューア上に表示する

図 2-1 3DCityDB の概要

2.2. 3DCityDB の作成

3DCityDB が公開している Docker イメージを使用して、3DCityDB の挙動確認を実施した。挙動確認結果については、本節以降に示す。

なお、Docker イメージを使用して 3DCityDB の環境を構築すると、データベースサーバーや 3DCityDB のデータベーススキーマをセットアップすることなく、すぐに 3DCityDB を使用することが可能である。

今回の挙動確認では、PostgreSQL を使用した 3DCityDB の Docker イメージによるデータベース作成と、Importer/Exporter の Docker イメージによるデータベースへの CityGML データの入出力の動作確認を実施した。（https://github.com/tum-gis/3dcitydb-docker-postgis、https://hub.docker.com/u/3dcitydb）

初めに、3DCityDB の Docker イメージを使用して、3DCityDB の環境構築を行った結果を示す。なお、環境構築は root 権限を持つユーザーで行うこと。

【OS 情報】

AlmaLinux release 9.2

【環境構築手順】

1.docker コマンドの代替として podman をインストールする

dnf -y install podman

2.3DCityDB の Docker イメージを pull する

podman pull 3dcitydb/3dcitydb-pg

・pull 対象の選択肢が表示された場合は以下を選択する

docker.io/3dcitydb/3dcitydb-pg:latest

・取得した Docker イメージの確認方法

podman images

3.Pod の作成

podman pod create -p 5432:5432 --name 3dcitydb_pod

3DCityDB では、3DCityDB に対してデータをインポート/エクスポートするための Docker イメージが別途用意されており、後述するデータのインポート/エクスポート機能の動作確認で使用する。そのため、複数のコンテナ間で通信が可能な Pod を作成する。

4.コンテナの起動

podman run -d --pod 3dcitydb_pod --name 3dcitydb -e POSTGRES_PASSWORD=pass -e SRID=6697 3dcitydb/3dcitydb-pg

POSTGRES_PASSWORD : PostgreSQL のパスワード（入力内容はユーザー任意）

SRID : データベースに保存するデータの座標系（EPSG コード）

5.データベーステーブルの確認

podman exec -it 3dcitydb /bin/bash←コンテナ内にアクセス

psql -h localhost -p 5432 -U postgres -d postgres←データベースに接続

¥dt←テーブルの一覧表示

データベース内のテーブルの確認が終了したら、データベースとの接続を切り、コンテナ内から抜ける。

¥q←データベースとの接続を切断

exit←コンテナ内から抜ける

```
[root@localhost vagrant]# podman exec -it 3dcitydb /bin/bash
root@beffd2c18dcf:/3dcitydb# psql -h localhost -p 5432 -U postgres -d postg
psql (14.6 (Debian 14.6-1.pgdg110+1))
Type "help" for help.

postgres=# \dt
                List of relations
  Schema  |           Name           | Type  |  Owner
----------+--------------------------+-------+----------
 citydb   | address                  | table | postgres
 citydb   | address_to_bridge        | table | postgres
 citydb   | address_to_building      | table | postgres
 citydb   | ade                      | table | postgres
 citydb   | aggregation_info         | table | postgres
 citydb   | appear_to_surface_data   | table | postgres
 citydb   | appearance               | table | postgres
 citydb   | breakline_relief         | table | postgres
 citydb   | bridge                   | table | postgres
 citydb   | bridge_constr_element    | table | postgres
 citydb   | bridge_furniture         | table | postgres
 citydb   | bridge_installation      | table | postgres
 citydb   | bridge_open_to_them_srf  | table | postgres
 citydb   | bridge_opening           | table | postgres
 citydb   | bridge_room              | table | postgres
 citydb   | bridge_thematic_surface  | table | postgres
 citydb   | building                 | table | postgres
 citydb   | building_furniture       | table | postgres
 citydb   | building_installation    | table | postgres
 citydb   | city_furniture           | table | postgres
 citydb   | citymodel                | table | postgres
          途中省略
 tiger    | state_lookup             | table | postgres
 tiger    | street_type_lookup       | table | postgres
 tiger    | tabblock                 | table | postgres
 tiger    | tabblock20               | table | postgres
 tiger    | tract                    | table | postgres
 tiger    | zcta5                    | table | postgres
 tiger    | zip_lookup               | table | postgres
 tiger    | zip_lookup_all           | table | postgres
 tiger    | zip_lookup_base          | table | postgres
 tiger    | zip_state                | table | postgres
 tiger    | zip_state_loc            | table | postgres
 topology | layer                    | table | postgres
 topology | topology                 | table | postgres
(103 rows)
```

図 2-2 3DCityDB のテーブル一覧

2.3. データのインポート

3DCityDB が公開している Importer/Exporter ツールでは以下のフォーマットのファイルを読み込むことが可能である。

表 2-2 入力フォーマット

フォーマット	サポートバージョン	備考
CityGML (*.gml、*.xml)	ver.2.0、ver. 1.0、ver.0.4	
CityJSON (*.JSON、*.cityJSON)	ver.1.0.x	
gzip (*.gz、*.gzip)	-	CityGML 又は CityJSON ファイルの圧縮ファイル
ZIP (*.zip)	-	CityGML 又は CityJSON ファイルの圧縮ファイル

Importer/Exporter の Docker イメージを使用して、前節で作成した 3DCityDB に対して、CityGML データのインポートを行った。なお、インポートデータは Project PLATEAU が公開している CityGML ファイル（https://www.geospatial.jp/ckan/dataset/plateau）を使用した。

以下に作業手順を示す。

【インポート作業手順】

1. Importer/Exporter の Docker イメージを pull する

```
# podman pull 3dcitydb/impexp
```

・ pull 対象の選択肢が表示された場合は以下を選択する

docker.io/3dcitydb/impexp:latest

2. CityGML のインポート

```
# podman run -it --rm --name impexp --pod 3dcitydb_pod -v /vagrant_data:/vagrant_data 3dcitydb/impexp import -H 127.0.0.1 -d postgres -u postgres /vagrant_data/13100_tokyo23-ku_2022_citygml_1_2_op/udx/bldg/53392633_bldg_6697_2_op.gml
```

上記コマンドの太字部分がインポート設定であり、最後に記載している CityGML ファイルパスを変更すると他の入力データを読み込むことが可能である。なお、Importer/Exporter ツールが入力ファイルにアクセスするためには、ホスト OS（AlmaLinux）とコンテナ間においてディレクトリを共有する必要がある。そのため、podman run -v オプションを使用して、ホスト OS の任意ディレクトリをコンテナの任意ディレクトリにマウントしている。

```
# podman run -v [ホスト OS のディレクトリ]:[コンテナのディレクトリ]
```

3.インポートデータの確認

```
# podman exec -it 3dcitydb /bin/bash←コンテナ内にアクセス
# psql -h localhost -p 5432 -U postgres -d postgres←データベースに接続
# select * from building;←sql 文でデータ確認
```

インポートデータが格納されていることを確認できたら、データベースとの接続を切り、コンテナ内から抜ける。

```
# ¥q←データベースとの接続を切断
# exit←コンテナ内から抜ける
```

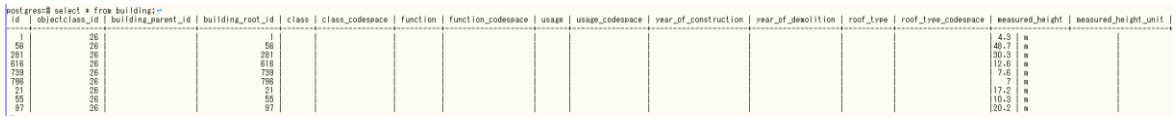


図 2-3 インポートデータの確認結果

今回は、Docker イメージを利用した関係上 CLI での動作確認を行ったが、Importer/Exporter ツールには GUI も用意されている。

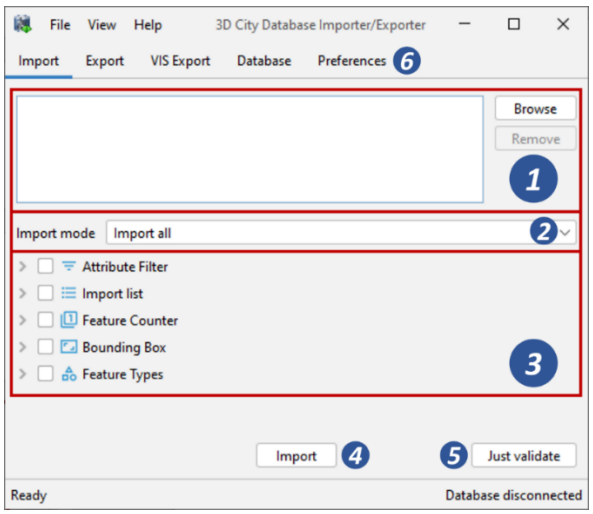


図 2-4 Import の GUI

3DCityDB, 3dcitydb-docs (2023) Fig.4.33 The import dialog. 引用

2.4. データのエクスポート

Importer/Exporter ツールには、CityGML 及び CityJSON を出力可能なエクスポート機能と ArcGIS などで読み込み可能な KML、COLLADA、glTF を出力可能なエクスポート機能の 2 種類が存在する。3DCityDB のドキュメントでは、前者を Export、後者を Visualization Export として記載している。

エクスポート機能の出力フォーマットを以下に示す。

表 2-3 Export 機能の出力フォーマット

フォーマット	サポートバージョン	備考
CityGML (*.gml、*.xml)	ver.2.0、ver. 1.0	
CityJSON (*.JSON、*.cityJSON)	ver.1.0.x	
gzip (*.gz、*.gzip)	-	挙動確認では、CityGML ファイルの圧縮ファイルを出力することを確認
ZIP (*.zip)	-	挙動確認では、CityGML ファイルの圧縮ファイルを出力することを確認

表 2-4 Visualization Export 機能の出力フォーマット

フォーマット	参考
KML	Wilson, T. (2008): OGC® KML, OGC® Standard Version 2.2.0. Open Geospatial Consortium, Doc. No. 07-147r2, April 14th. Weblink (accessed March 2020): https://portal.opengeospatial.org/files/?artifact_id=27810
COLLADA	Barners, M., Finch, E. L. (2008): COLLADA - Digital Asset Schema Release 1.5.0. The Khronos Group Inc., Sony Computer Entertainment Inc, April 2008. Weblink (accessed March 2020): https://www.khronos.org/files/collada_spec_1_5.pdf
glTF	glTF - Efficient, Interoperable Transmission of 3D Scenes and Models, Khronos, Weblink (accessed March 2020): https://www.khronos.org/gltf

Export の動作確認として、Importer/Exporter の Docker イメージを用いて 3DCityDB に登録済みのデータを CityGML ファイルにエクスポートする作業を行った。

以下に作業手順を示す。

【Export の動作確認手順】

1. Importer/Exporter の Docker イメージを pull する。

※　インポート作業で実施済みの場合は不要

```
# podman pull 3dcitydb/impexp
```

・　pull 対象の選択肢が表示された場合は以下を選択する。

docker.io/3dcitydb/impexp:latest

2. データのエクスポート

```
# podman run -it --rm --name impexp --pod 3dcitydb_pod -v /vagrant_data:/vagrant_data 3dcitydb/impexp export -H 127.0.0.1 -d postgres -u postgres -p -o /vagrant_data/export/output.gml
```

※　データベースのパスワード入力が必要

（3DCityDB の Docker コンテナを起動する際に podman run コマンドに入力した POSTGRES_PASSWORD の入力値）

上記コマンドの太字部分がエクスポート設定であり、-o オプションで指定する出力ファイルパスの拡張子によって出力フォーマットを判断している。上記コマンドでは、CityGML（*.gml）を指定した。なお、Importer/Exporter ツールが出力先にアクセスするためには、ホスト OS（AlmaLinux）とコンテナ間においてディレクトリを共有する必要がある。そのため、podman run -v オプションを使用して、ホスト OS の任意ディレクトリをコンテナの任意ディレクトリにマウントしている。

```
# podman run -v [ホスト OS のディレクトリ]:[コンテナのディレクトリ]
```

図 2-5 エクスポート結果（CityGML の記載内容）

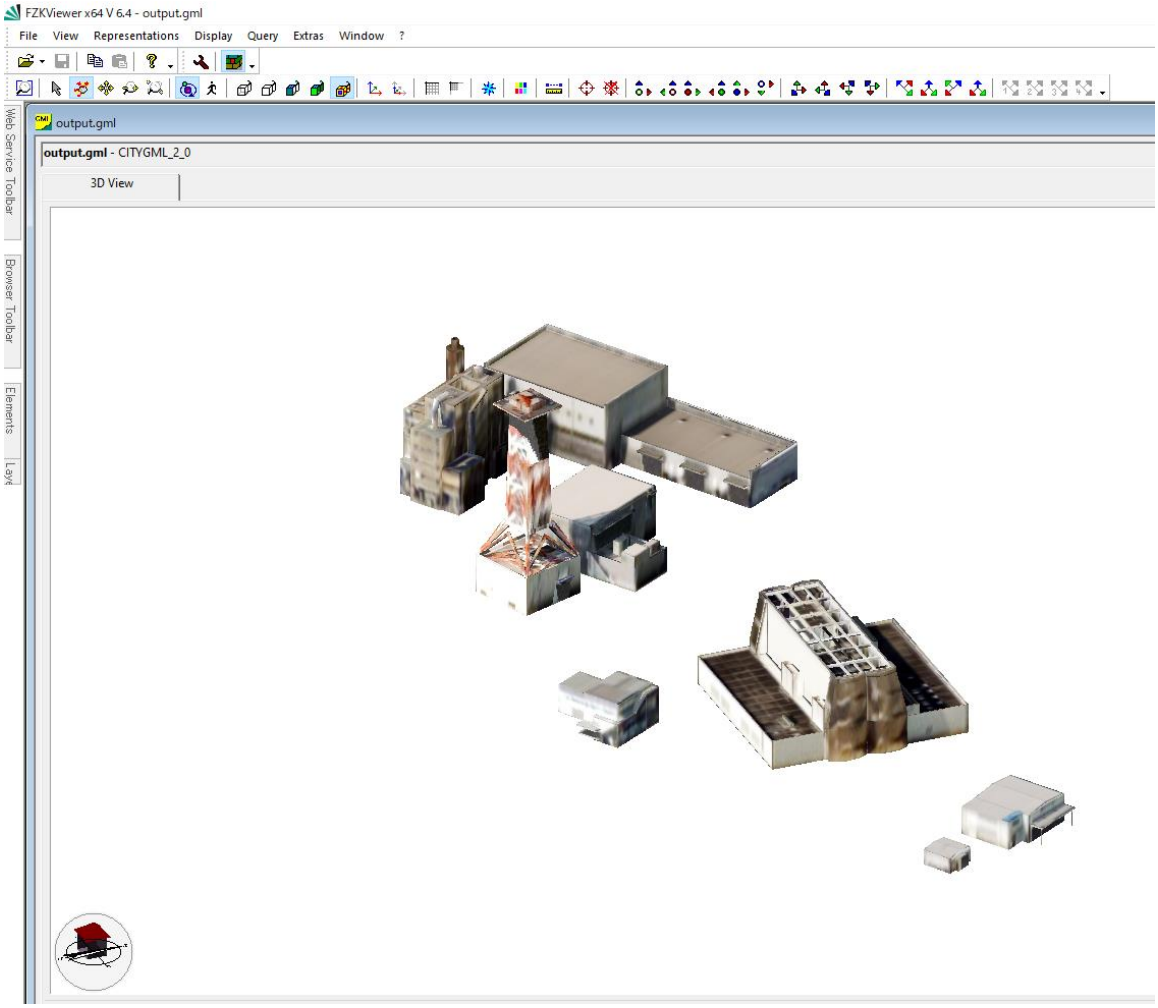


図 2-6 エクスポート結果（LOD2 表示）

3DCityDB にインポートした CityGML データとエクスポート結果の CityGML データを可視化し、表示されるモデルを比較したところ、エクスポートデータはインポートデータと同一のモデルが出力されていることを確認した。3DCityDB が提供している Importer/Exporter ツールのインポート機能では、 Project PLATEAU が公開している CityGML ファイルで使用している i-UR 2.0 に関する属性が読み込めないため、インポートとエクスポートした CityGML データに必ず属性の差異が発生する。そのため、属性データの比較は実施していない。

下図に出力ファイルパスの拡張子を json に変更し、CityJSON ファイルを出力した結果を示す。出力したモデルデータは CityGML 出力と同一のため省略する。

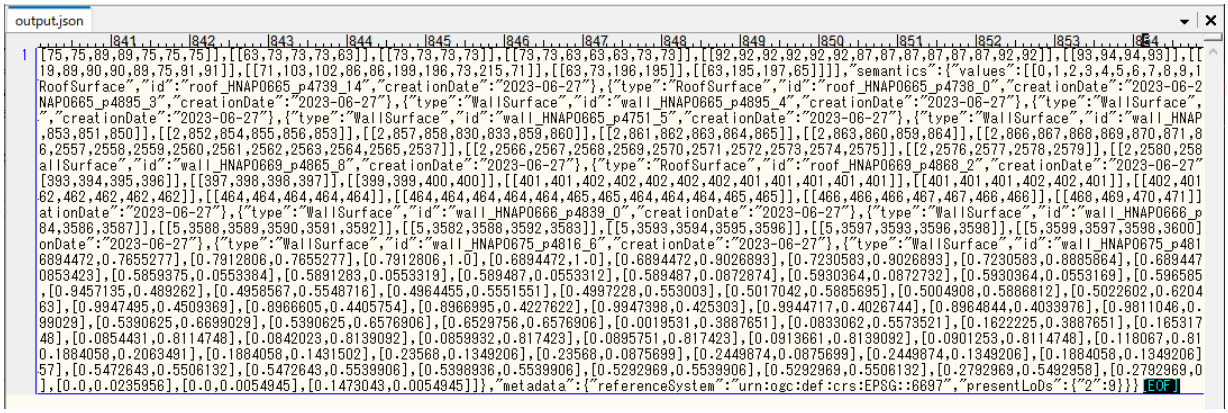


図 2-7 エクスポート結果(CityJSON の記載内容)

Visualization Export の動作確認として、Export の動作確認と同様に 3DCityDB のデータを KML ファイルにエクスポートする作業を行った。

【Visualization Export の動作確認手順】

1. データのエクスポート

```
# podman run -it --rm --name impexp --pod 3dcitydb_pod -v
/vagrant_data:/vagrant_data 3dcitydb/impexp export-vis -H 127.0.0.1 -d postgres -
u postgres -p -l 2 -D geometry -t Building -o /vagrant_data/export/vis/output.kml
```

※ データベースのパスワード入力が必要
（3DCityDB の Docker コンテナを起動する際に podman run コマンドに入力した POSTGRES_PASSWORD の入力値）

※ Export 同様、ホスト OS とコンテナ間のディレクトリ共有のために podman run -v オプションによってマウント設定を行っている。

上記コマンドの太字部分がエクスポート設定である。オプションの簡易説明を以下に記載する。

【オプションコマンド】

-l ： モデルの LOD 設定
0 - 4、halod (highest available LOD) を設定可能

-D ： 表示形式設定
- LOD とは別にモデルの表示方法を決定するオプション。
- collada、geometry、extruded、footprint から 1 つ又は複数を選択可能。
- 各表示形式は、指定された LOD 内の都市オブジェクトのジオメトリに基づいて生成される。

-g ： タイル分割設定
- 行列数を指定する方法と、タイル幅を指定する方法がある。
- タイル幅を指定する方法の場合は、タイル幅を用いてタイル数を算出する際に丸めが発生する場合は、指定したタイル幅より小さな値となる可能性がある。

-t ： タイプ設定

- 出力したい機能型名（Building、Road、LandUse 等）を 1 つ以上カンマ区切りで列挙可能。
- 機能型名は、公式の CityGML 機能型名又は CityGML ADE の機能型名と一致する必要がある。

-o ： 出力ファイルパス

メイン出力となる KML ファイルパスを指定する。

図 2-8 表示形式

3DCityDB, 3dcitydb-docs (2023) Fig. 4.79 The same building displayed as Footprint, Extruded, Geometry, COLLADA/glTF with textures (from top left to bottom right) 引用

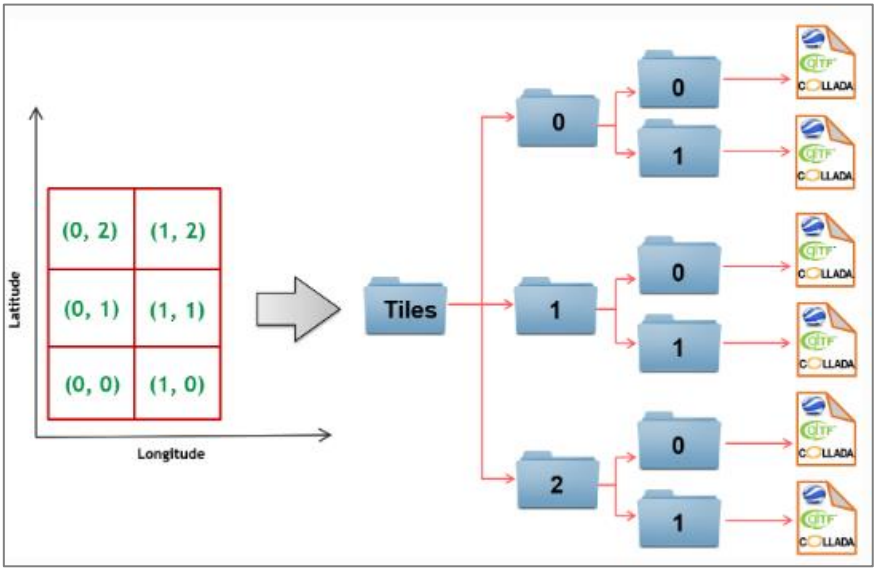


図 2-9 タイル分割時の出力フォルダ構成

3DCityDB, 3dcitydb-docs (2023)Fig. 4.80 Example: hierarchical directory structure for export of 3x2 tiles 引用

エクスポート結果を以下に示す。

Tile フォルダ内にモデルデータが出力されているため、KMZView において KML ファイルを読み込み、出力モデルの確認を行った。ビューア固有の問題のためか、出力モデルの壁面が描画されていないが、出力モデルの表示位置や屋根の形状等より入力 CityGML と同一のモデルが出力されていると考える。

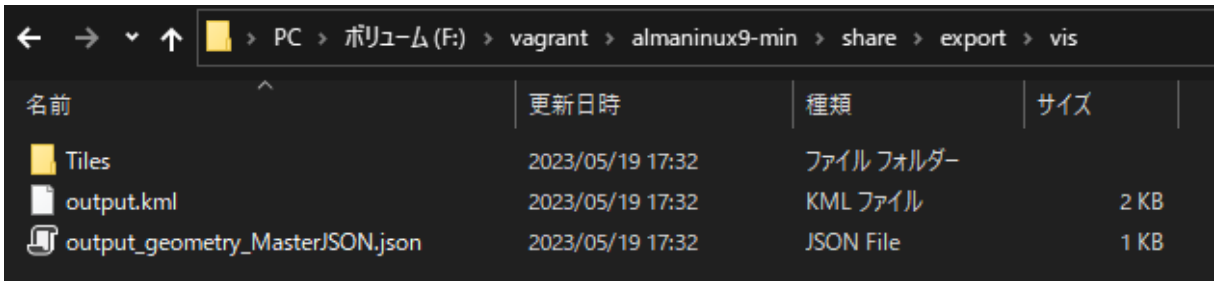


図 2-10 Visualization Export 結果

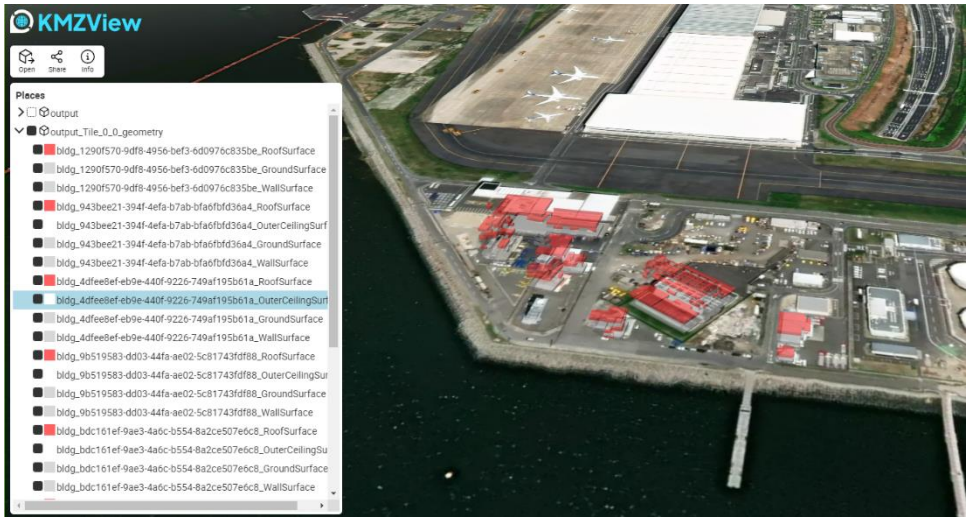


図 2-11 エクスポートした KML ファイルを表示した結果

エクスポート時の表示形式設定を COLLADA/glTF に変更して、データのエクスポートを行った結果、COLLADA 形式でデータが出力された。 Importer/Exporter の公式ドキュメント（ https://3dcitydb-docs.readthedocs.io/en/latest/impexp/export-vis.html ）では、表示形式設定が COLLADA/glTF の場合の出力フォーマットは、「COLLADA and/or glTF」と表記されているため、今回の動作結果のように COLLADA 形式のファイルのみを出力し、glTF 形式のファイルが出力されない場合がある。

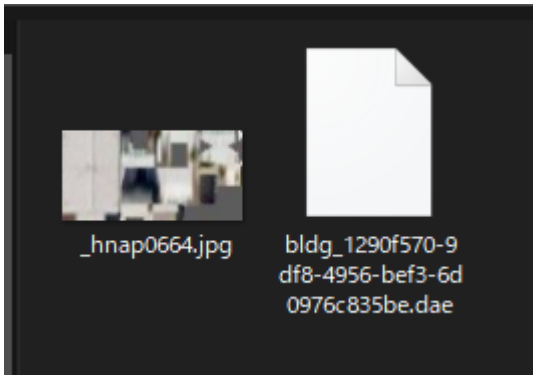


図 2-12 COLLADA/glTF モードでのエクスポート結果

今回は、Docker イメージを利用した関係上 CLI での動作確認を行ったが、Importer/Exporter ツールには、Export と Visualization Export の GUI も用意されている。

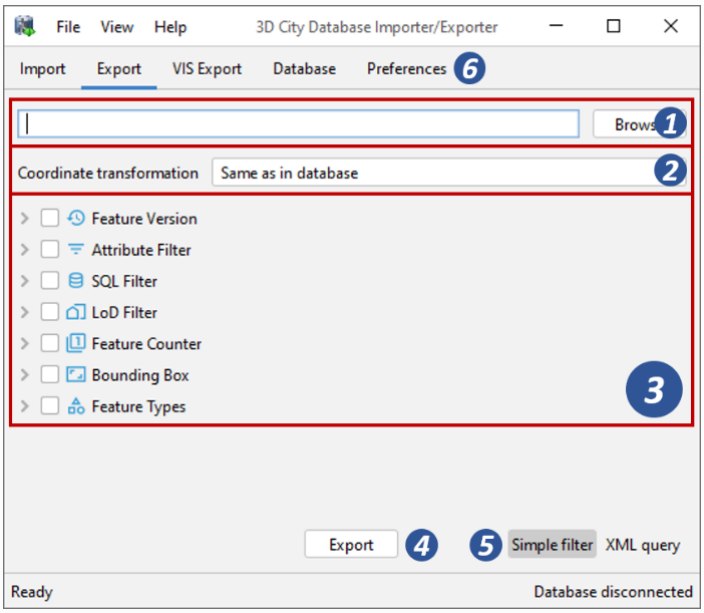


図 2-13 Export の GUI

3DCityDB, 3dcitydb-docs (2023)

Fig. 4.53 The export dialog. 引用

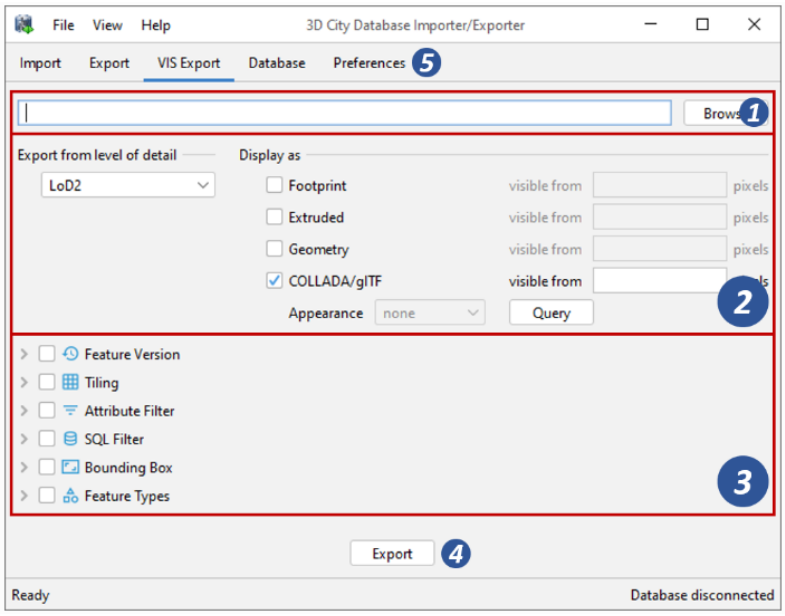


図 2-14 Visualization Export の GUI

3DCityDB, 3dcitydb-docs (2023)

Fig. 4.78 The VIS Export tab allowing for exporting visualization models from the 3DCityDB.

2.5. Importer/Exporter のプラグイン

Importer/Exporter ツールにプラグインを追加することで、機能の拡張が可能である。3DCityDB では以下のプラグインを公開している。

- Spreadsheet Generator Plugin
  3DCityDB に保存されている都市オブジェクトの属性データを CSV ファイル、又は Microsoft Excel ファイル (*.xlsx) にエクスポート可能。
- ADE Manager Plugin
  CityGML の拡張規則 (ADE) を、3DCityDB 用のスキーマに自動的に変換し、既存の 3DCityDB インスタンスと ADE スキーマの登録を解除して、再登録が可能。

なお、独自プラグインを開発することも可能であり、Spreadsheet Generator Plugin、ADE Manager Plugin のソースコードをテンプレートとして使用可能である。開発には Java を使用する。

2.6. データベース拡張検討

調査時の 3DCityDB が対応している CityGML 2.0 は、OGC（Open Geospatial Consortium）が 3D 都市モデルの国際標準として策定したデータ形式である。この CityGML 2.0 は、基本的な地物及び属性が用意されているが、汎用的な地物や属性の使用に制限がないため、都市の特性やユースケースに応じてデータ形式を拡張することが可能である。データ形式の拡張には ADE(Application Domain Extention）を使用して、CityGML の仕様自体を拡張し、地物や属性の応用スキーマを新たに定義することができる。Project PLATEAU では、都市計画情報等に着目した ADE である「i-UR」を用いて CityGML の仕様を拡張しており、標準規格である CityGML 2.0 に対応している 3DCityDB では i-UR に該当する地物や属性の管理が行えない状態である。3DCityDB で、i-UR に該当する地物や属性を管理するためには、データベーススキーマを変更して i-UR の地物や属性をデータベースに保存できるようにし、Importer/Exporter ツールにて i-UR に該当する地物や属性のインポートとエクスポートを行えるようにする必要がある。3DCityDB では、ADE による CityGML の仕様拡張に対応するために、Importer/Exporter ツールに対して「ADE Manager Plugin」が用意されている。ADE Manager Plugin では、CityGML の仕様拡張を定義している XSD（XML Schema definition）ファイルを使用して標準規格である CityGML 2.0 のデータベーススキーマを XSD ファイルに記載された仕様に自動で拡張することが可能である。なお、データベーススキーマの拡張は容易に行えるが、Importer/Exporter ツールの ADE 対応は、ADE で拡張した地物や属性をインポーポート又はエクスポート可能なようにライブラリを開発する必要がある。Importer/Exporter ツールは Java で作成されており、ライブラリは jar ファイルとして用意する必要がある。本検証では、Importer/Exporter ツールの ADE Manager Plugin を使用して、標準規格の 3DCityDB を i-UR 2.0 に拡張する作業を行った。また、i-UR 2.0 に拡張したデータベースに対して、PLATEAU で公開されている i-UR 2.0 対応済みの CityGML データ

（https://www.geospatial.jp/ckan/dataset/plateau）が Importer/Exporter ツールのインポート機能を使用して読み込み可能であるか確認を行った。

今回は、i-UR 2.0 データを読み込めるようにするための Importer/Exporter ツールのライブラリ作成までは行わない。そのため、インポート機能の挙動確認では、i-UR 2.0 データを無視して標準の CityGML データを読み込めるのか、全てのデータが読み込み不可であるのかを確認した。

2.6.1. 検討環境

検討環境として、VirtualBox 上に仮想環境を作成した。今回は、Importer/Exporter ツールの GUI 機能を使用するため、仮想 OS に AlmaLinux （Server with GUI）を使用する。

なお、3DCityDB の使用には、PostgreSQL、PostGIS、及び Java が必要となるため、下表に示すバージョンのものをインストールした。

3DCityDB に関しては、公式サイトにツール一式をまとめた 3dcitydb-suite が用意されていたため、最新版（v2022.2.0）を使用した。

表 2-5 検討環境

項目	ツール	備考
仮想環境	VirtualBox 7.0.4	
仮想 OS	AlmaLinux 9.2 (Server with GUI)	
DB	PostgreSQL ver.15.3、PostGIS ver.3.3	
Java	OpenJDK ver.11.0.19	
3DCityDB	3dcitydb-suite v2022.2.0 (https://github.com/3dcitydb/3dcitydb-suite/releases/tag/v2022.2.0)	・3D City Database ver.4.4.0 ・Importer/Exporter ver.5.3.0 ・Spredsheet Generator Plugin ver.4.2.0 ・ADE Manager Plugin ver.2.2.0 ・3D Web Map Client ver.1.9.1 ・Web Feature Service ver.5.3.0

2.6.2. データベース拡張

仮想 OS に、空の 3DCityDB の作成と、Importer/Exporter ツールのインストールを行った後に、i-UR 2.0 の XSD ファイルを使用して、データベースを拡張するための拡張パッケージの作成を実施した。

なお、i-UR 2.0 の XSD ファイルは、G 空間情報センターの CKAN(https://www.geospatial.jp/iur/)から入手したファイルを使用している。

i-UR 2.0 の XSD ファイルは 4 ファイルあり、Importer/Exporter ツールの ADE Manager において拡張パッケージを一括で作成するために、4 ファイルを取りまとめた XSD ファイルを作成した。

UrbanRevitalizationADE.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="https://www.geospatial.jp/iur/2.0" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="https://www.geospatial.jp/iur/2.0"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="2.0">
  <xs:import namespace="https://www.geospatial.jp/iur/urf/2.0" schemaLocation="schemas/urbanFunction.xsd"/>
  <xs:import namespace="https://www.geospatial.jp/iur/urg/2.0" schemaLocation="schemas/statisticalGrid.xsd"/>
  <xs:import namespace="https://www.geospatial.jp/iur/uro/2.0" schemaLocation="schemas/urbanObject.xsd"/>
  <xs:import namespace="https://www.geospatial.jp/iur/urt/2.0" schemaLocation="schemas/publicTransit.xsd"/>
</xs:schema>
```

図 2-15 i-UR 2.0 の XSD ファイルをまとめるための XSD ファイル

【拡張パッケージの作成方法】

1. XML Schema (XSD) に作成した XSD ファイルを指定する
2. Read XML Schema ボタンを押下すると、XML Namespace 欄に i-UR 2.0 の名前空間が表示される
3. XML Namespace 欄において、拡張パッケージを作成する名前空間を選択する
4. 拡張パッケージに必要な情報を記載する
5. Output folder に拡張パッケージの出力先を指定する
6. Transform ボタンを押下する

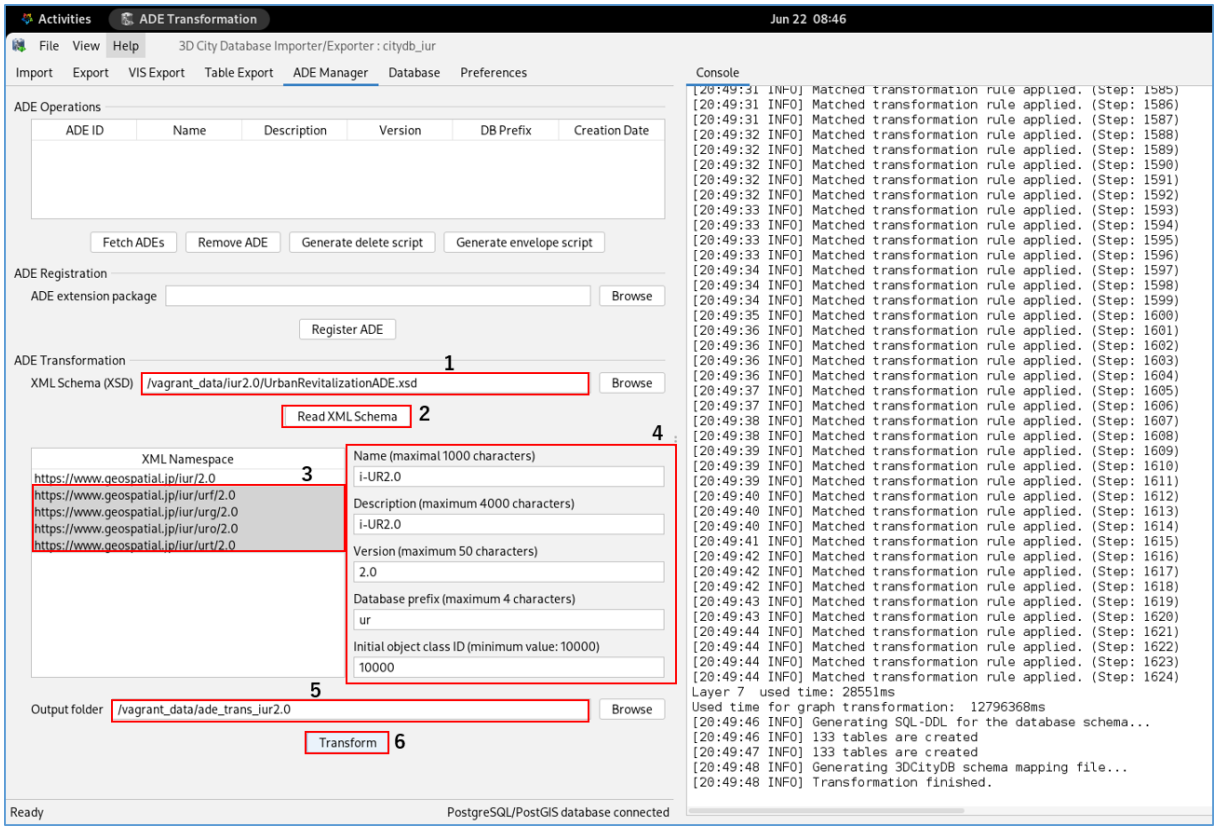


図 2-16 拡張パッケージの作成

```
ade_trans_iur2.0
    ├ 3dcitydb
    │   ├oracle
    │   │   ├ CREATE_ADE_DB.sql
    │   │   ├ DISABLE_ADE_VERSIONING.sql
    │   │   ├ DROP_ADE_DB.sql
    │   │   └ ENABLE_ADE_VERSIONING.sql
    │   └ postgreSQL
    │       ├ CREATE_ADE_DB.sql
    │       └ DROP_ADE_DB.sql
    │
    └ schema-mapping
        └ schema-mapping.xml
```

図 2-17 出力した拡張パッケージのファイル構成

3DCityDB に作成した拡張パッケージを適用して、i-UR 2.0 のデータを保持可能なデータベースに拡張する。拡張パッケージの適用方法を以下に示す。

【拡張パッケージの適用】

1. ADE extension package に出力した拡張パッケージのフォルダパス（拡張パッケージ作成時の Output folder に指定したフォルダパス）を指定する
2. Register ADE ボタンを押下する

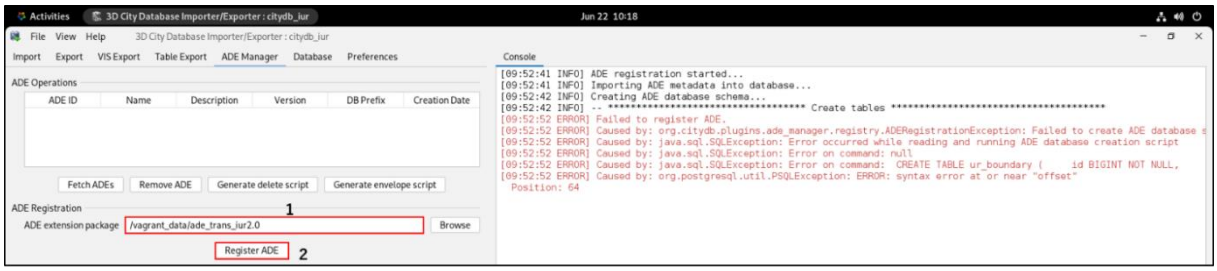


図 2-18 拡張パッケージの適用

拡張パッケージ作成で生成したパッケージを、そのまま使用してデータベースの拡張を行うと、テーブルのフィールド名である offset が、PostgreSQL の予約語である offset と重複してしまい、エラーになる。そのため、今回は拡張パッケージの CREATE TABLE 文を修正してから、データベース拡張を実施した。

- 拡張パッケージで生成された SQL 文では、ur_boundary テーブルの作成においてエラー（suntax error）が発生した。
- PostgreSQL では、「offset」は予約語のためフィールド名に使用できないことが原因だった。
- PostgreSQL を用いる場合、カラム名を変更した上で拡張を行い、データ参照時に考慮する必要がある。
- 「offset」のカラム名を暫定的に変更することで ur_boundary テーブルの作成ができることを確認した。

  ※予約語：SQL 文で使用するキーワード。テーブルや列名の名前には使用不可。

  ※PostgreSQL では、「offset」は予約語に該当するが、Oracle では予約語に該当しない。

ade_trans_iur2.0/3dcitydb/PostgreSQL/CREATE_ADE_DB.sql

【修正前】

```
-- ------------------------------------
-- ur_boundary
-- ------------------------------------
CREATE TABLE ur_boundary
(
    id BIGINT NOT NULL,
    offset NUMERIC,
    offset_uom VARCHAR(1000),
    offsetdirection VARCHAR(1000),
    zone_boundary_id BIGINT,
    PRIMARY KEY (id)
);
```

【修正後】

```
-- ------------------------------------
-- ur_boundary
-- ------------------------------------
CREATE TABLE ur_boundary
(
    id BIGINT NOT NULL,
    offset_val NUMERIC,
    offset_uom VARCHAR(1000),
    offsetdirection VARCHAR(1000),
    zone_boundary_id BIGINT,
    PRIMARY KEY (id)
);
```

図 2-19 CREATE_ADE_DB.sql の修正

拡張パッケージの適用が完了すると、Importer/Exporter ツールの Database タブの ADEs タブにおいて適用されている拡張パッケージの状態を確認することが可能である。今回作成した拡張パッケージはデータベースの拡張のみであり、Importer/Exporter ツールのライブラリ修正は行っていないため、ADEs タブの Database 欄はチェック済みであるが、Importer/Exporter 欄は×印となっている。

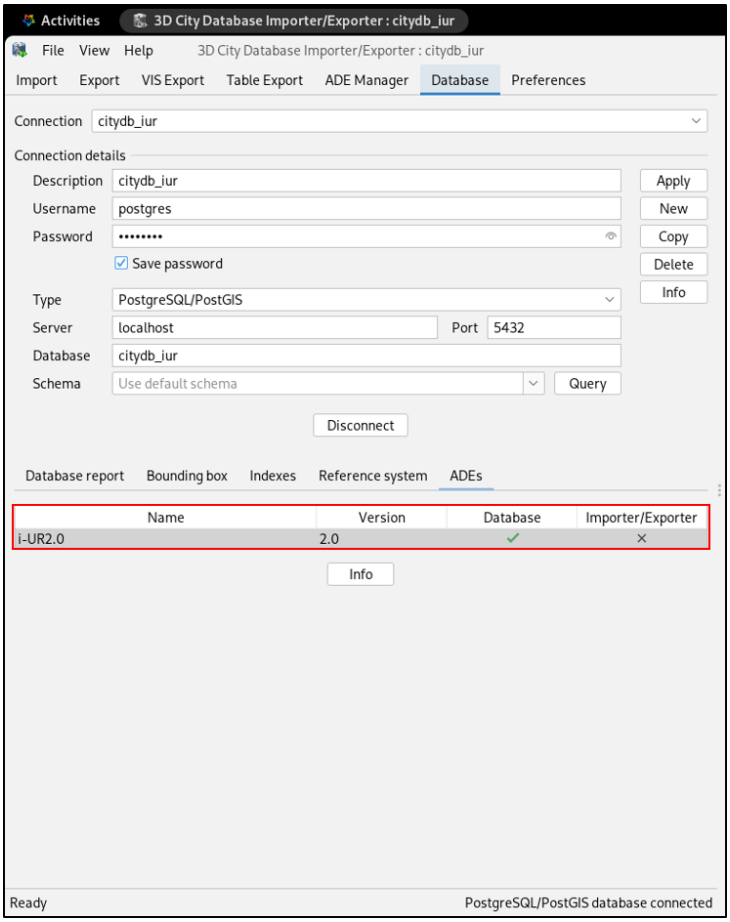


図 2-20 適用済み拡張パッケージの確認

実際のデータベースのテーブルの変化状況については下図のとおりである。

左:i-UR 2.0 拡張前（66 テーブル）

右:i-UR 2.0 拡張後（199 テーブル）

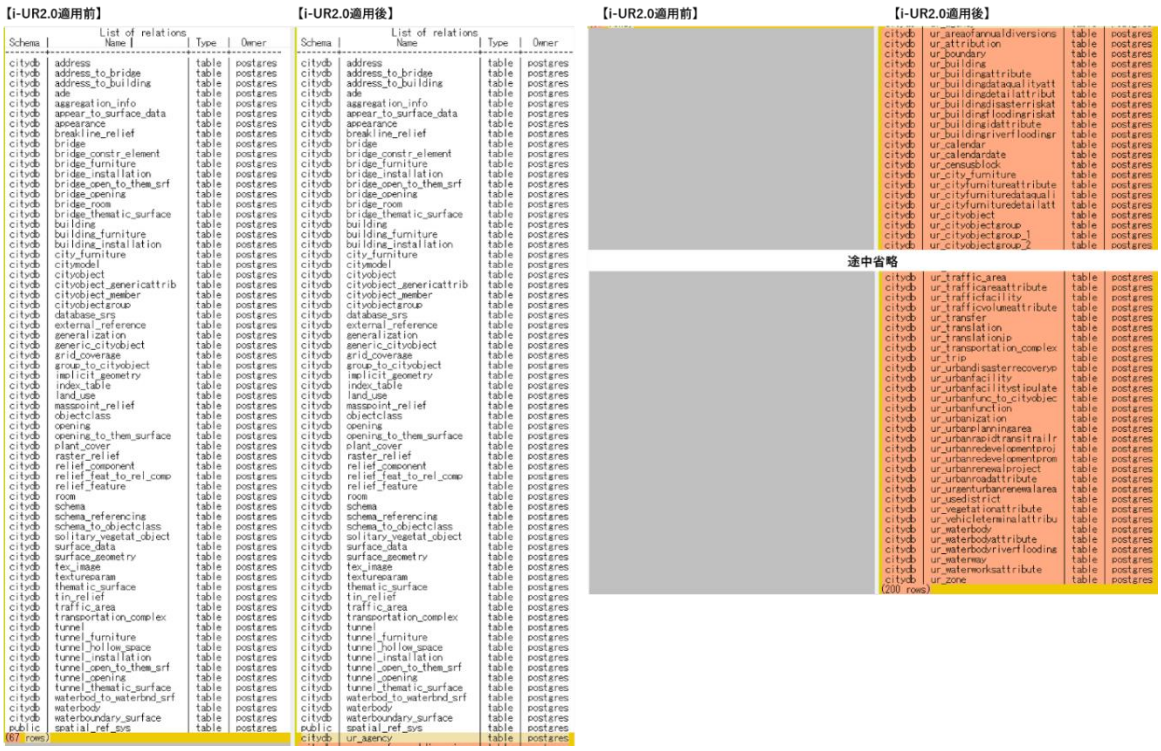


図 2-21 拡張パッケージ適用前後のデータベーステーブル差分

建物（Building）の ER 図で i-UR 2.0 に拡張されていることを確認した。

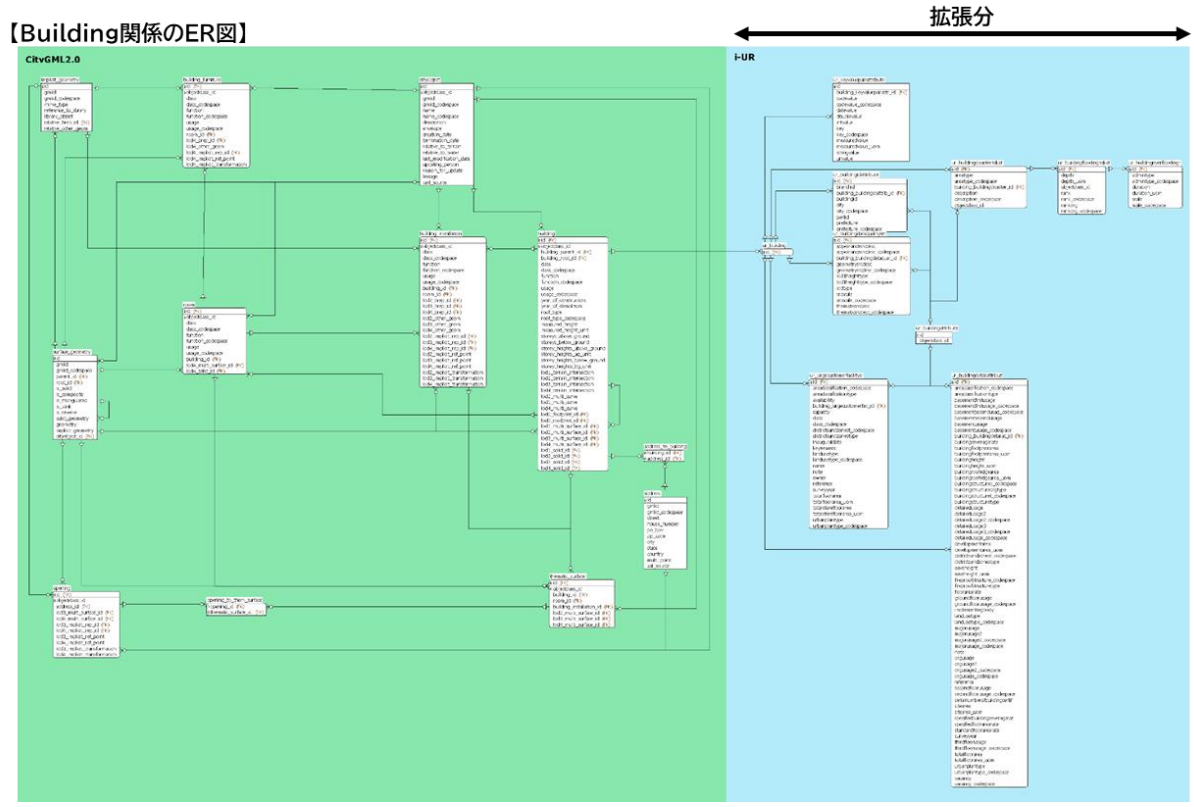


図 2-22 拡張パッケージ適用前後の ER 図差分

2.6.3. CityGML データのインポート

前項で作成した i-UR 2.0 拡張済みのデータベースに対して、PLATEAU に公開されている i-UR 2.0 対応済みの CityGML データ（https://www.geospatial.jp/ckan/dataset/plateau）のインポートを行った。今回は i-UR 2.0 データを読み込めるようにするための Importer/Exporter ツールのライブラリ作成を行っていないため、i-UR 2.0 データを無視して標準の CityGML データを読み込めるかどうかを確認することが本作業の目的である。

CityGML データのインポート方法とインポート後の各テーブルの登録レコード数を以下に示す。

【CityGML ファイルのインポート】

1. Browse ボタンを押下して、インポートする CityGML ファイルを選択する
2. import ボタンを押下して、CityGML ファイルを読み込む

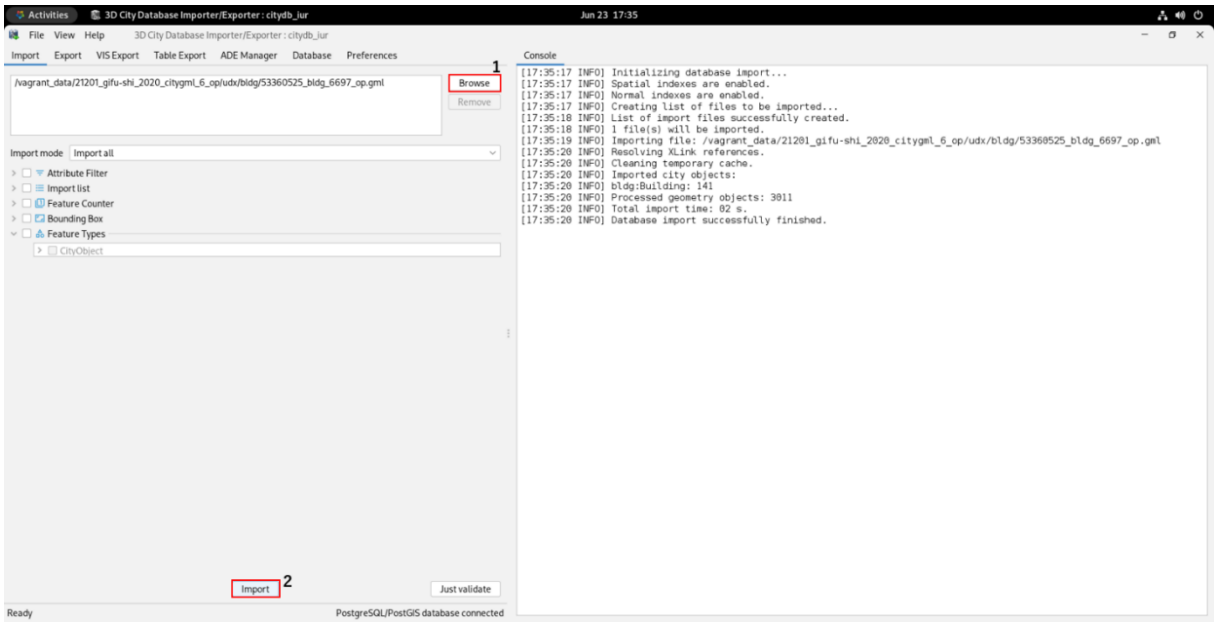


図 2-23 CityGML のインポート（GUI）

図 2-24 インポートファイルの内容

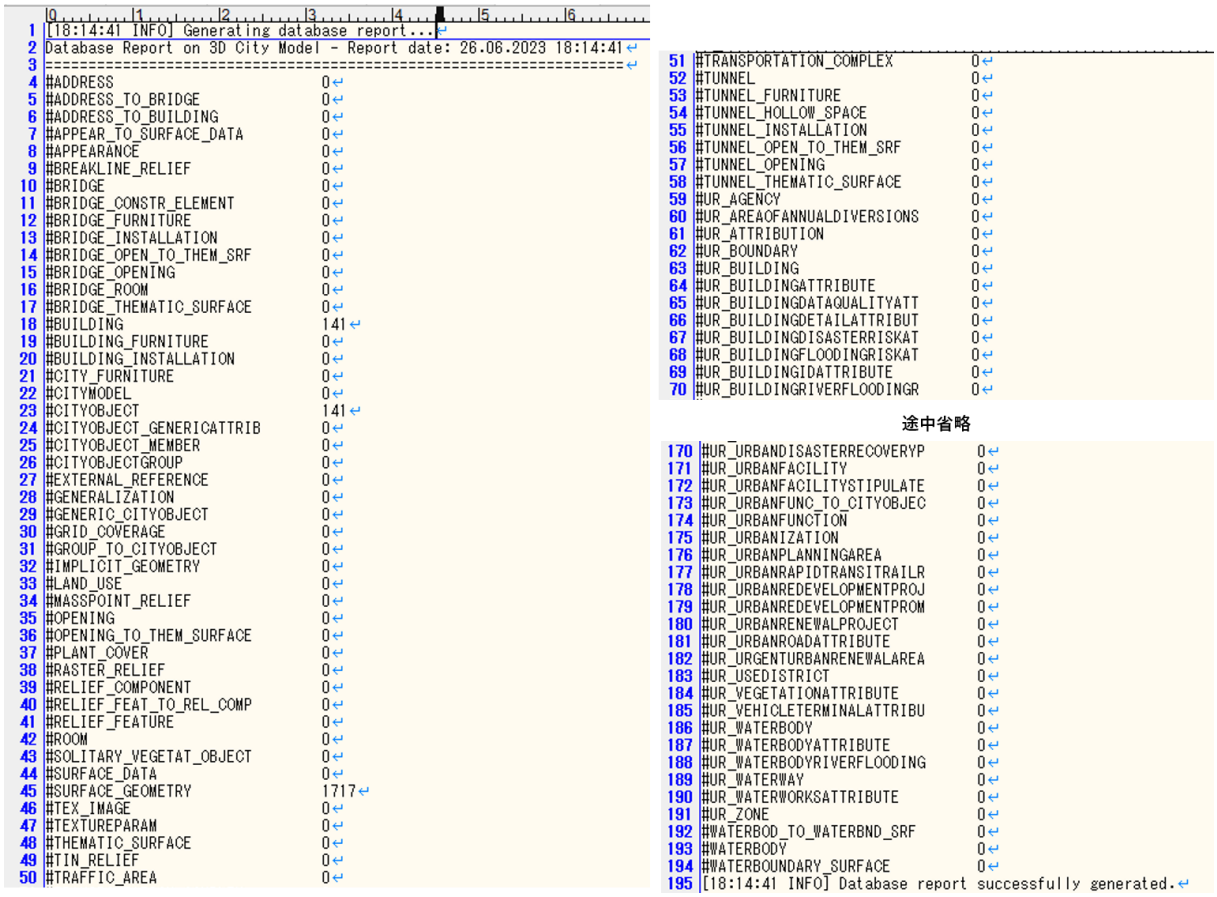


図 2-25 CityGML インポート後のレコード登録数

今回は建物の CityGML データのインポートを行った。上図のレコード登録数を確認すると、BUILDING テーブル等の建物に関連するテーブルにレコードが登録されていることが分かる。また、i-UR 2.0 拡張パッケージによって追加された接頭辞が「UR」のテーブルに関しては登録レコードがないことが分かる。入力 CityGML データには i-UR の記載があるが、データベースには i-UR に関連するレコードが登録されていないため、i-UR 2.0 データを無視して標準の CityGML データのみをインポートしていることが分かる。

2.7. 3DCityDB と Cesium との連携

3DCityDB と Cesium との連携については、既に 3DCityDB から 2 つを連携した 3DCityDB-Web-Map-Client というソフトウェアパッケージが公開されているため連携可能である。(https://github.com/3dcitydb/3dcitydb-web-map)

3DCityDB-Web-Map-Client は、Cesium Virtual Globe をベースに開発されており、任意サイズの 3D 都市モデルを高性能に 3D で可視化し、インタラクティブに探索することができるツールである。

3DCityDB-Web-Map-Client では、ユーザーが 3D 都市モデルを便利に閲覧や探索が可能なように、Cesium Virtual Globe に拡張機能を追加している。主な追加機能は、3DCityDB がエクスポートする KML/glTF モデルを 3DCityDB-Web-Map-Client 上で直接視覚化する機能である。この追加機能に

よって、3DCityDB と Cesium は KML/glTF モデルを利用したデータ連携が可能となっている。なお、地物の属性情報に関しては、Spreadsheet Generator Plugin 機能（3DCityDB の Importer/Exporter のプラグイン）が出力するスプレッドシートを用いてデータ連携が可能となっている。スプレッドシートによる連携の他に、使用しているデータベースがサポートしている RESTful API（PostgreSQL ならば、PostgREST）を用いた連携方法も存在する。

RESTful API による連携では、直接 3DCityDB から属性情報を参照することが可能である。

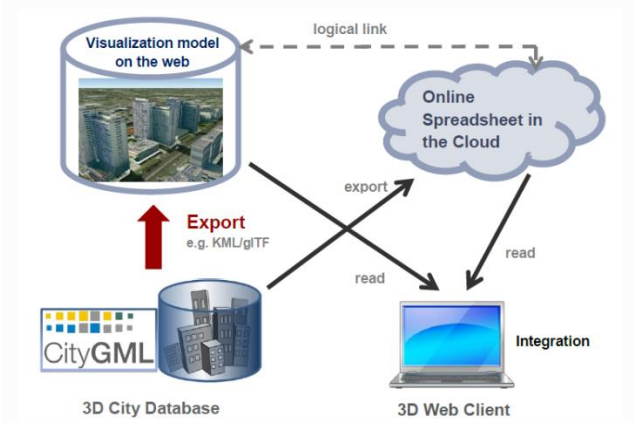
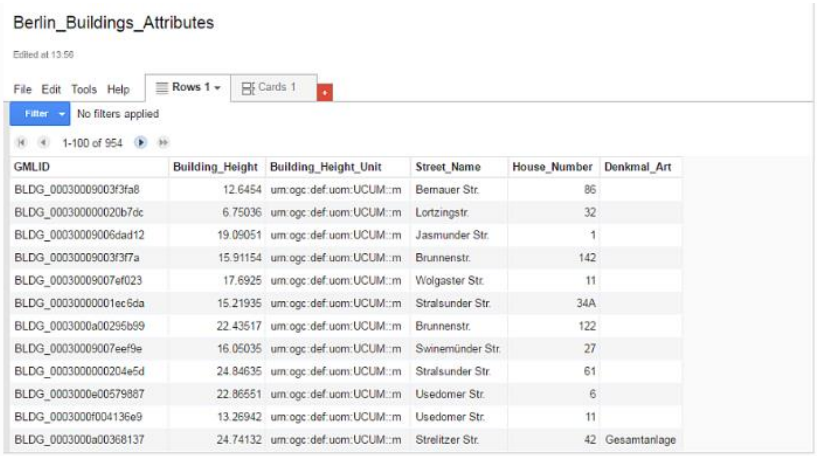


図 2-26 3DCityDB と Cesium とのデータ連携

3DCityDB, 3dcitydb-docs (2023) Fig. 6.8 Coupling an online spreadsheet with a 3D visualization model (i.e. a KML/glTF visualization model) in the cloud


Berlin_Buildings_Attributes

Edited at 13:56

File Edit Tools Help Rows 1 Cards 1

Filter No filters applied

1-100 of 954

GMLID	Building_Height	Building_Height_Unit	Street_Name	House_Number	Denkmal_Art
BLDG_00030009003f3fa8	12.6454	urn:ogc:def:uom:UCUM::m	Bernauer Str.	86	
BLDG_000300000020b7dc	6.75036	urn:ogc:def:uom:UCUM::m	Lortzingstr.	32	
BLDG_00030009006dad12	19.09051	urn:ogc:def:uom:UCUM::m	Jasmunder Str.	1	
BLDG_00030009003f3f7a	15.91154	urn:ogc:def:uom:UCUM::m	Brunnenstr.	142	
BLDG_00030009007ef023	17.6925	urn:ogc:def:uom:UCUM::m	Wolgaster Str.	11	
BLDG_00030000001ec6da	15.21935	urn:ogc:def:uom:UCUM::m	Stralsunder Str.	34A	
BLDG_0003000a00295b99	22.43517	urn:ogc:def:uom:UCUM::m	Brunnenstr.	122	
BLDG_00030009007eef9e	16.05035	urn:ogc:def:uom:UCUM::m	Swinemünder Str.	27	
BLDG_000300000204e5d	24.84635	urn:ogc:def:uom:UCUM::m	Stralsunder Str.	61	
BLDG_0003000e00579887	22.86551	urn:ogc:def:uom:UCUM::m	Usedomer Str.	6	
BLDG_0003000f004136e9	13.26942	urn:ogc:def:uom:UCUM::m	Usedomer Str.	11	
BLDG_0003000a00368137	24.74132	urn:ogc:def:uom:UCUM::m	Strelitzer Str.	42	Gesamtanlage

図 2-27 スプレッドシート例

3DCityDB, 3dcitydb-docs (2023) Fig. 6.9 Example of an online spreadsheet

2.7.1. 3DCityDB-Web-Map-Client の機能

3DCityDB-Web-Map-Client の起動画面と各コントロールの概要を以下に示す。

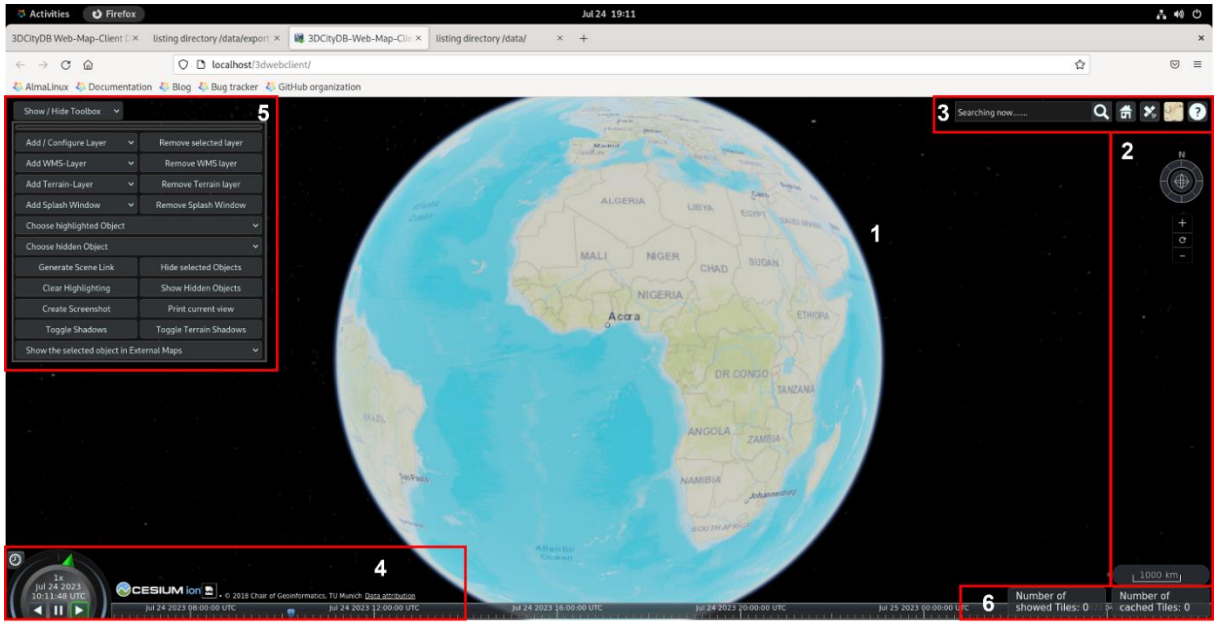


図 2-28 3DCityDB-Web-Map-Client

1. 3D グローブ

メインビューとなる地球儀では、マウスやタッチスクリーンを使用して、カメラの視点を変更しながら地図を閲覧可能である。なお、地球儀の背景は、「3. ツールキット」内の背景レイヤーの選択から変更することが可能である。

3DCityDB からエクスポートした都市モデルや、WMS（Web Map Service）、DTM（Digital Terrain Model）を追加表示することが可能である。

2. ナビゲーションコンポーネント

上部のコンパスとナビゲーター（ズームイン/ズームアウト）はカメラの視点を制御するコントロールである。マウスやタッチスクリーンによるカメラ視点制御と同等な操作を行うことが可能である。

下部は、距離スケールを表示している。

図 2-29 ナビゲーションコンポーネント

3. ツールキット

ツールキットは、以下の 5 種類の機能を有する。

図 2-30 ツールキット

・ ジオコーダー

ジオコーダーのルーペボタンを押下すると検索ボックスが展開され、[経度]、[緯度]形式による明示的な位置、又は特定の場所を検索するための住所名による場所の検索が可能である。場所が特定されると、カメラの視点が自動で特定位置に移動する。

・ ホーム

ホームボタンを押下すると、カメラの視点がデフォルト状態に戻る。

・　ジオロケーション

ジオロケーションボタンを押下すると、ユーザーの現在地にカメラの視点を移動する。

動作環境がモバイルの場合は、ユーザーの現在地を表示する機能（Location snapshot）、ユーザーの現在地を定期的に表示するリアルタイムトラッキング機能（Real-time orientation tracking）、一人称視点で現在地をトラッキング表示する機能（First-person view）を有する。

図 2-31 モバイル端末の場合のジオロケーション

3DCityDB, 3dcitydb-docs (2023) Fig. 6.31 From left to right, the 3 modes of geolocation-based features: Location snapshot, Real-time orientation tracking and First-person view

図 2-32 一人称視点

3DCityDB, 3dcitydb-docs (2023)

Fig. 6.32 Real-time orientation tracking and First-person View on mobile devices

・　背景レイヤー選択

背景レイヤー選択では、数に示す画像や地形データを選択することで地球儀の背景レイヤーを変更することが可能である。デフォルトデータとして、Bing Maps、OpenStreetMap、EsriMaps などが提供している画像レイヤー等を用意している。

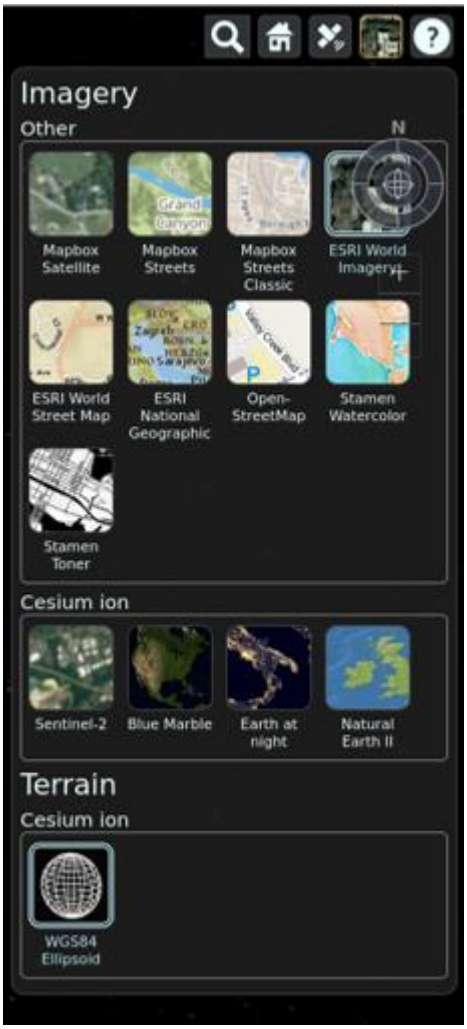


図 2-33 背景レイヤー選択

・　ヘルプ

ヘルプでは、メインビューとなる 3D グローブの操作方法の簡易説明、及び 3DCityDB-Web-Map-Client についてのインフォメーションを記載されている。

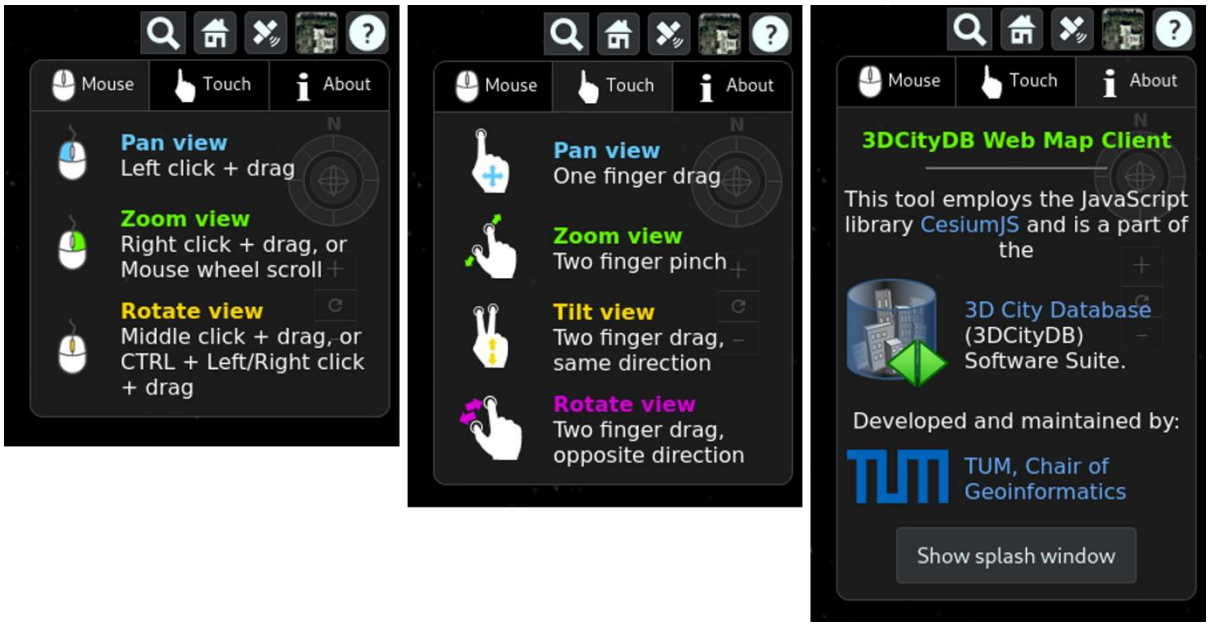


図 2-34 ヘルプ

4.　クレジットコンテナ

3DCityDB-Web-Map-Client の開発と使用に関与したソフトウェアとデータプロバイダーに関するクレジットを表記している。

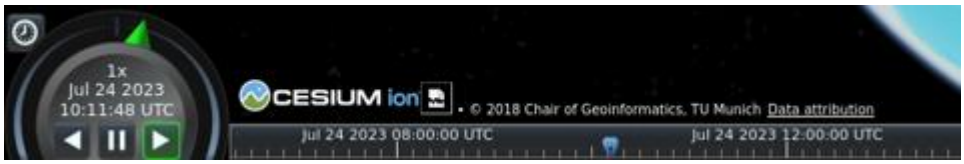


図 2-35 クレジットコンテナ

5.　ツールボックス

ツールボックスは、ユーザーが入力するデータを制御するための拡張モジュールである。

【入力可能なデータ】

- KML/glTF モデル
- 属性データ（Google スプレッドシート）
- WMS （Web Map Service）
- デジタル地形データ

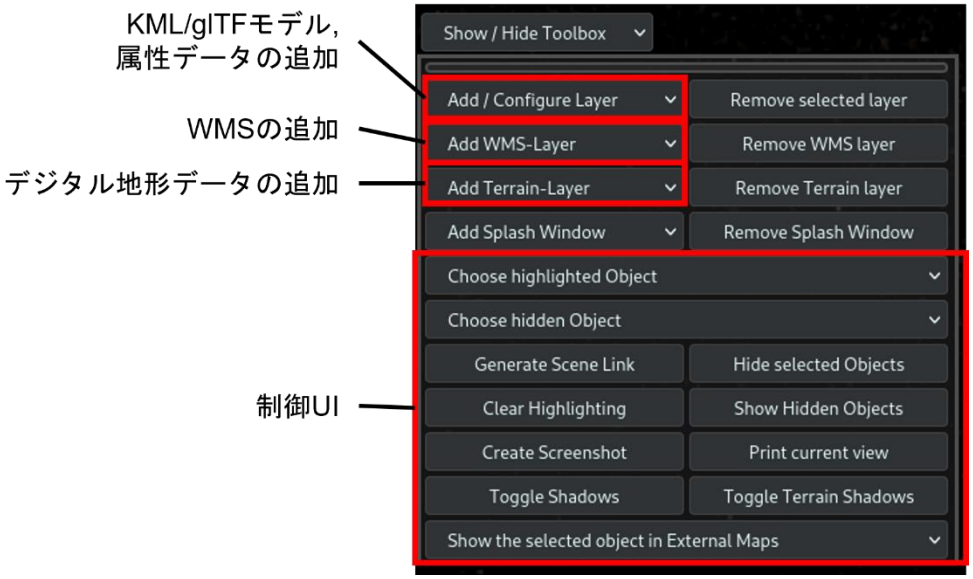


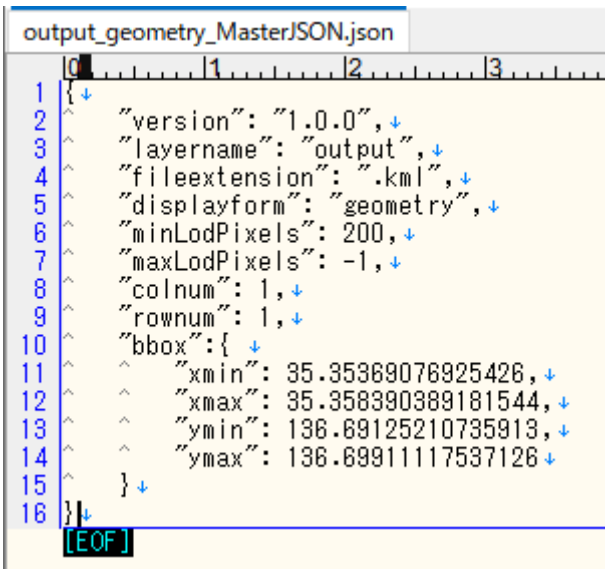
図 2-36 ツールボックス

本書では、3DCityDB からエクスポートした 3D 都市モデル（KML/glTF モデル、属性データ）の読み込みについて記載する。

3DCityDB からエクスポートした 3D 都市モデルを読み込むには、Add / Configure Layer 機能を使用する。なお、前提条件として、3DCityDB の Importer/Exporter ツールからエクスポートした 3D 都市モデルデータは、データサーバー上に保存されているものとする。

3D 都市モデルの読み込みでは、Add / Configure Layer 機能の URL 設定において、3DCityDB の Importer/Exporter ツールからエクスポートした 3D 都市モデルデータの JSON ファイル（*_MasterJSON.json）の URL を指定する必要がある。この JSON ファイルには、下図のようにデータ形式やデータ範囲などの 3D 都市モデルデータセットの設定が記載されている。

output_geometry_MasterJSON.json



```
{
    "version": "1.0.0",
    "layername": "output",
    "fileextension": ".kml",
    "displayform": "geometry",
    "minLodPixels": 200,
    "maxLodPixels": -1,
    "colnum": 1,
    "rownum": 1,
    "bbox":{
        "xmin": 35.35369076925426,
        "xmax": 35.358390389181544,
        "ymin": 136.69125210735913,
        "ymax": 136.69911117537126
    }
}
[EOF]
```

図 2-37 JSON ファイル（*_MasterJSON.json）例

また、属性データも併せて読み込む際は、thematicDataUrl に、属性情報が記載された Google スプレッドシートの URL を指定する必要がある。Name には登録レイヤー名を入力し、その他の設定項目に関しては入力データに合わせて入力値を設定する。

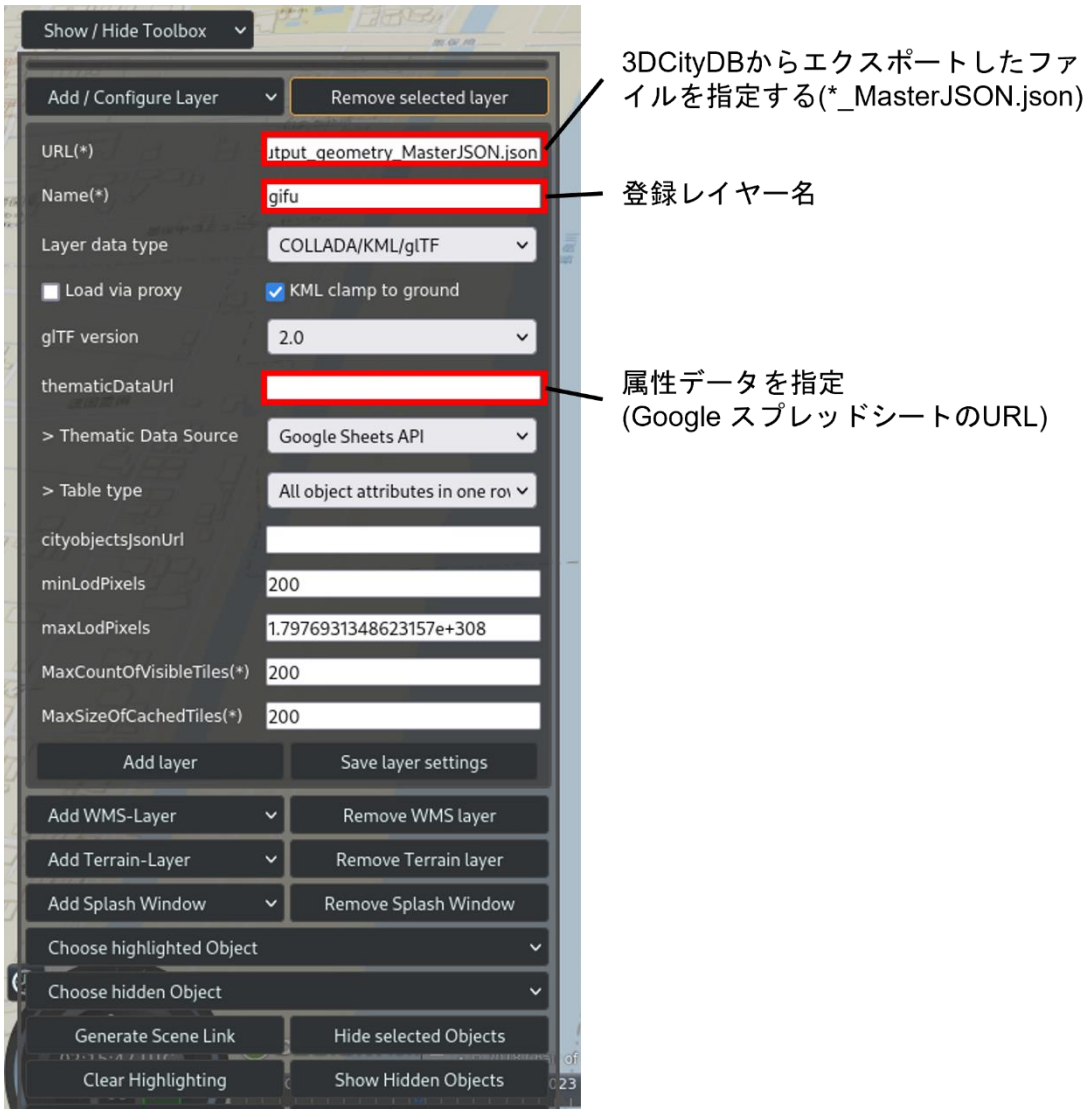


図 2-38 Add / Configure Layer

6. ステータス

3DCityDB-Web-Map-Client では、データサイズが大きい 3D 都市モデルを効率的に表示するためにタイリング方式を採用しており、画面上の表示サイズに応じて、解析や表示するタイルの選別を行っている。このタイリング方式では、キャッシュ機構をサポートしており、以前の処理でロードされたタイルデータを一時的にキャッシュに保存し、キャッシュからデータをロードすることで、リモートサーバーからデータをロードするよりも高速に表示することが可能である。

なお、キャッシュされたタイルデータ数が多ければ、より多くのメモリを消費しブラウザのメモリリークを引き起こす可能性がある。これを避けるために、ステータスによって、表示されたタイルデータとキャッシュされたタイルデータの量をリアルタイムで表示する。

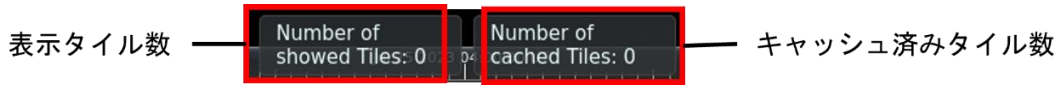


図 2-39 ステータス

タイルデータの設定は、Visualization Export 時に行うことが可能である。Tiling 設定からタイル幅を指定するか、タイルの行数と列数を指定することで、エクスポートデータのタイル数を制御可能である。また、visible from 設定にて各モデル（Footprint、Extruded、Geometry、COLLADA/glTF）の可視性を制御可能である。visible from 設定は正方形の辺のサイズを表し、データセットがビューアのスクリーンに投影される際に、この正方形よりも大きなスクリーン領域をデータセットが占めなければ可視化されない。Footprint と Extruded というように複数のモデルを同時にエクスポートする場合は、高解像度の visible from 設定が低解像度のモデルが不可視になるタイミングを定義する。この設定により、ビューア上でズームインやズームアウトした際に広域表示では詳細度の低いモデルを表示し、詳細表示では詳細度が高いモデルを表示するということが可能になる。

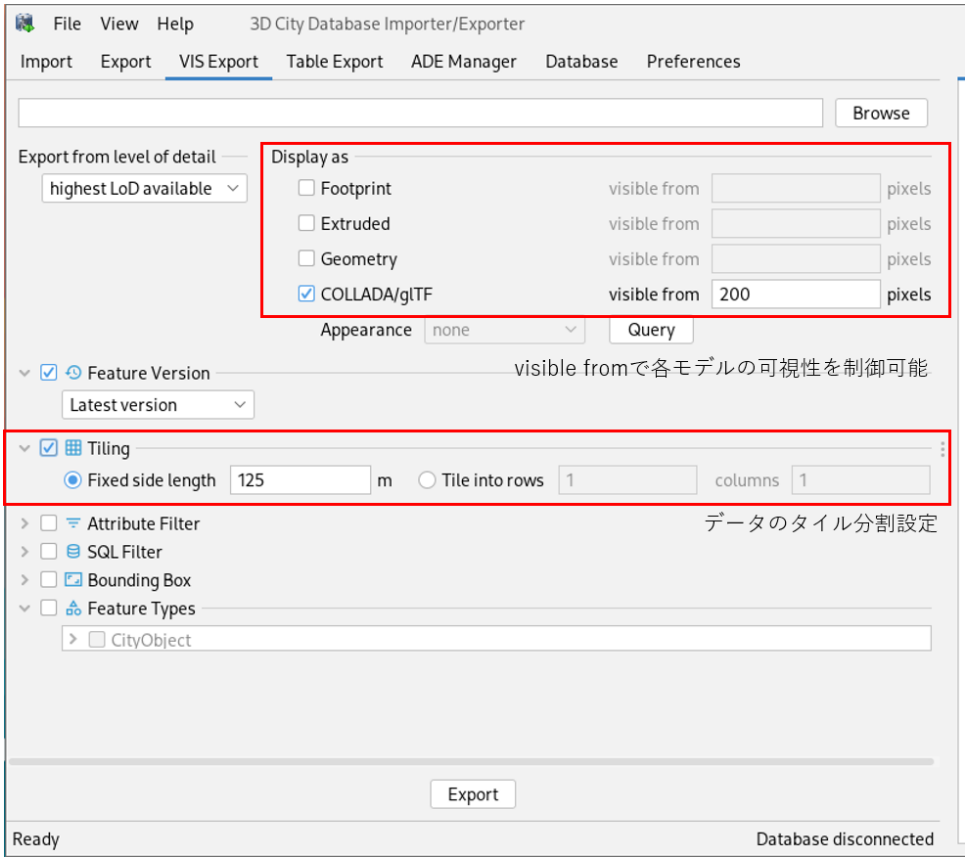


図 2-40 Visualization Export

2.7.2. 3DCityDB-Web-Map-Client と PostgREST の連携

3DCityDB-Web-Map-Client における、RESTful API を利用した属性情報の参照方法について記載する。本検討では、PostgreSQL ベースの 3DCityDB を使用している関係上、RESTful API として PostgREST を使用する。以下に、各ツールの関係図を示す。

図 2-41 3DCityDB-Web-Map-Client、PostgREST、3DCityDB の関係図

3DCityDB-Web-Map-Client から PostgREST を通して、3DCityDB の属性情報を参照するためには、3DCityDB に公開用の API の作成と、PostgREST から 3DCityDB へアクセスするためのロールを作成する必要がある。本検討では、建物モデル属性の公開用 API の作成を行った。公開用 API には、Importer/Exporter ツールにおいて出力した KML/glTF モデルデータとの関連付けを行うために cityobject テーブルの gmlid 属性を含む必要がある。また、公開する建物モデルの属性情報は、building テーブルの usage、year_of_construction、measured_height、storeys_above_ground 要素とした。

【3DCityDB の変更作業】

```
# DB に接続
psql -h localhost -p 5432 -U postgres -d citydb_v4
# 公開 API の作成
create view building_attr_view as select c.gmlid as gmlid, b.usage as usage, b.year_of_construction as year_of_construction, b.measured_height as measured_height, b.storeys_above_ground as storeys_above_ground from building as b join cityobject as c on b.id = c.id;
# ロール作成
create role web_anon nologin;
# web_anon に citydb スキーマのアクセスを許可
grant usage on schema citydb to web_anon;
# web_anon に作成した API の select 権限を付与
grant select on citydb.building_attr_view to web_anon;
# PostgREST からログインするためのロールを作成
create role authenticator noinherit login password 'password';
# authenticator に web_anon と同等の権限を付与
grant web_anon to authenticator;
```

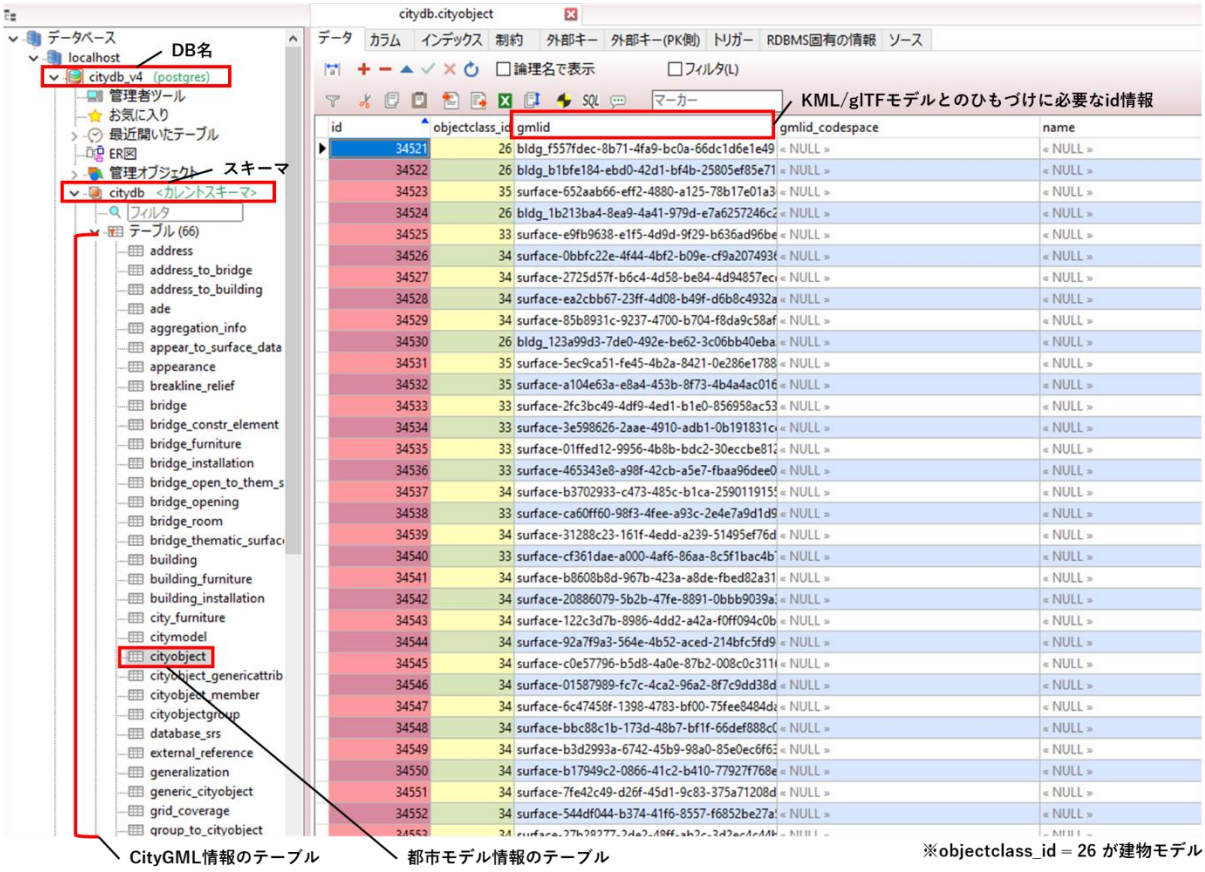


図 2-42 cityobject テーブル

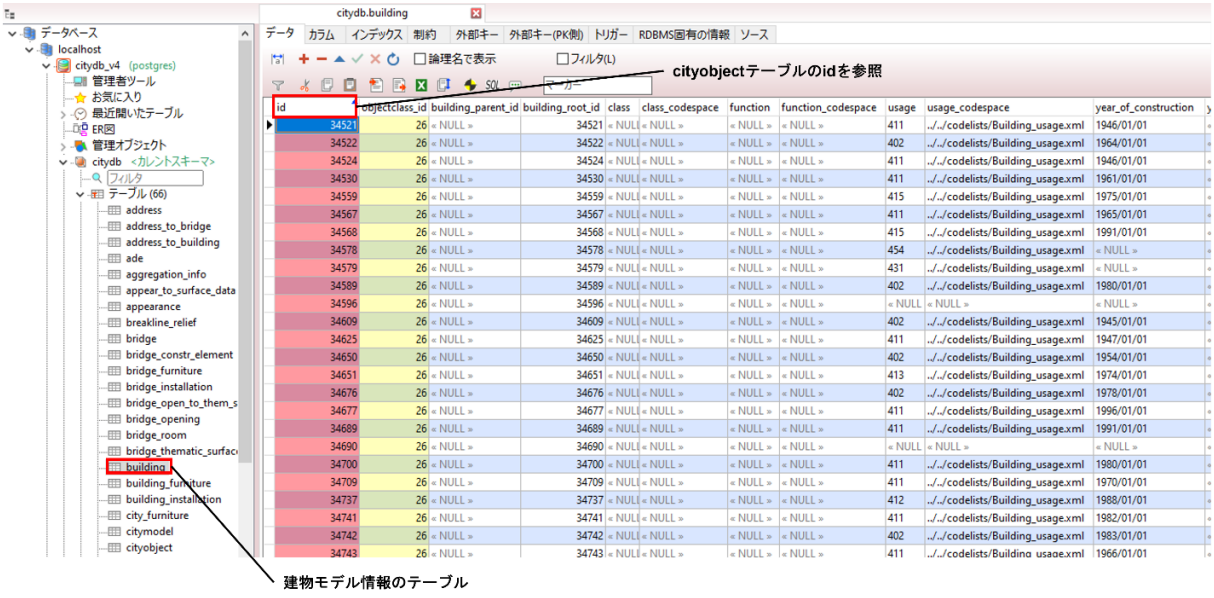
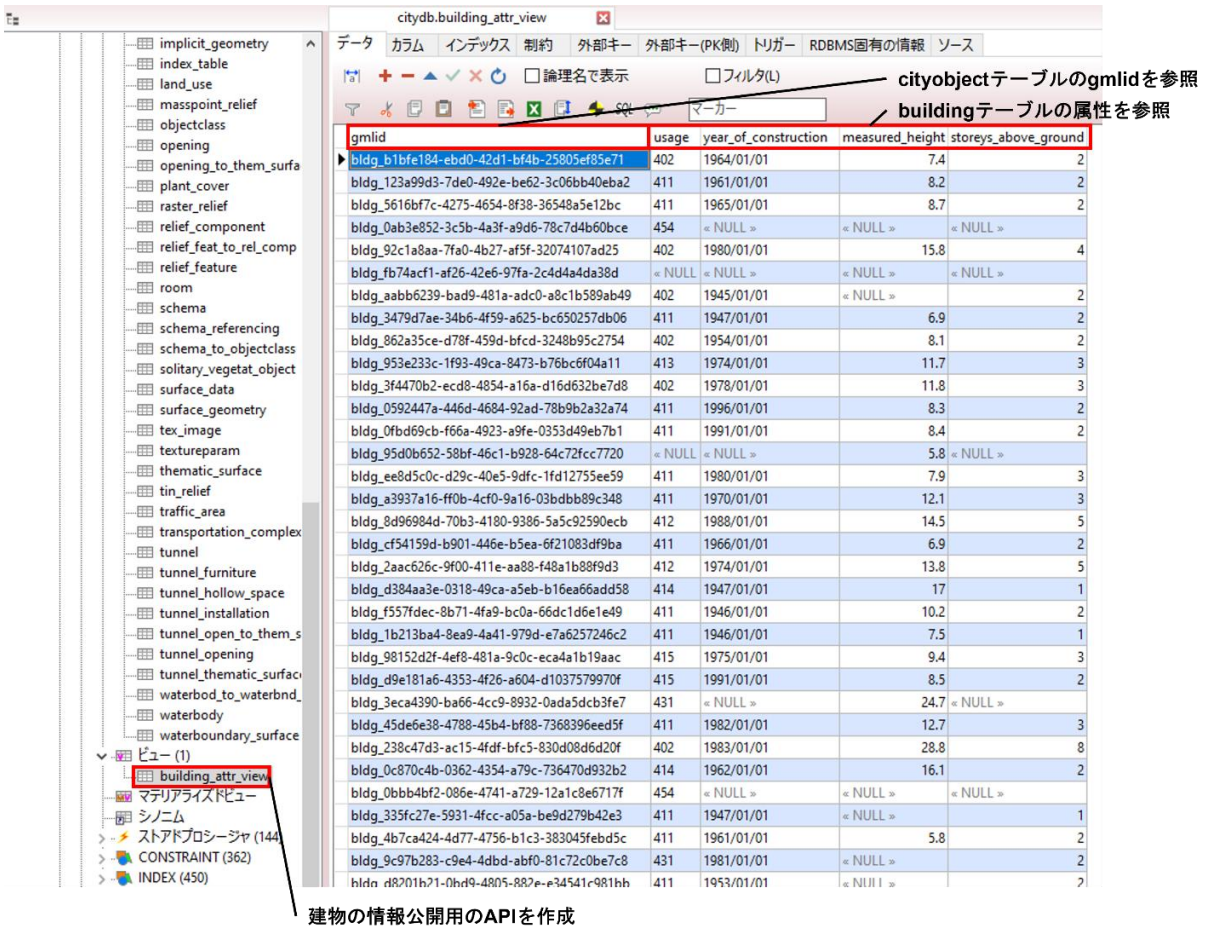


図 2-43 building テーブル



gmlid	usage	year_of_construction	measured_height	storeys_above_ground
bldg_b1bfe184-ebd0-42d1-bf4b-25805ef85e71	402	1964/01/01	7.4	2
bldg_123a99d3-7de0-492e-be62-3c06bb40eba2	411	1961/01/01	8.2	2
bldg_5616bf7c-4275-4654-8f38-36548a5e12bc	411	1965/01/01	8.7	2
bldg_0ab3e852-3c5b-4a3f-a9d6-78c7d4b60bce	454	« NULL »	« NULL »	« NULL »
bldg_92c1a8aa-7fa0-4b27-af5f-32074107ad25	402	1980/01/01	15.8	4
bldg_fb74acf1-af26-42e6-97fa-2c4d4a4da38d	« NULL	« NULL »	« NULL »	« NULL »
bldg_aabb6239-bad9-481a-adc0-a8c1b589ab49	402	1945/01/01	« NULL »	2
bldg_3479d7ae-34b6-4f59-a625-bc650257db06	411	1947/01/01	6.9	2
bldg_862a35ce-d78f-459d-bfcd-3248b95c2754	402	1954/01/01	8.1	2
bldg_953e233c-1f93-49ca-8473-b76bc6f04a11	413	1974/01/01	11.7	3
bldg_3f4470b2-ecd8-4854-a16a-d16d632be7d8	402	1978/01/01	11.8	3
bldg_0592447a-446d-4684-92ad-78b9b2a32a74	411	1996/01/01	8.3	2
bldg_0fbd69cb-f66a-4923-a9fe-0353d49eb7b1	411	1991/01/01	8.4	2
bldg_95d0b652-58bf-46c1-b928-64c72fcc7720	« NULL	« NULL »	5.8	« NULL »
bldg_ee8d5c0c-d29c-40e5-9dfc-1fd12755ee59	411	1980/01/01	7.9	3
bldg_a3937a16-ff0b-4cf0-9a16-03bdbb89c348	411	1970/01/01	12.1	3
bldg_8d96984d-70b3-4180-9386-5a5c92590ecb	412	1988/01/01	14.5	5
bldg_cf54159d-b901-446e-b5ea-6f21083df9ba	411	1966/01/01	6.9	2
bldg_2aac626c-9f00-411e-aa88-f48a1b88f9d3	412	1974/01/01	13.8	5
bldg_d384aa3e-0318-49ca-a5eb-b16ea66add58	414	1947/01/01	17	1
bldg_f557fdec-8b71-4fa9-bc0a-66dc1d6e1e49	411	1946/01/01	10.2	2
bldg_1b213ba4-8ea9-4a41-979d-e7a6257246c2	411	1946/01/01	7.5	1
bldg_98152d2f-4ef8-481a-9c0c-eca4a1b19aac	415	1975/01/01	9.4	3
bldg_d9e181a6-4353-4f26-a604-d1037579970f	415	1991/01/01	8.5	2
bldg_3eca4390-ba66-4cc9-8932-0ada5dcb3fe7	431	« NULL »	24.7	« NULL »
bldg_45de6e38-4788-45b4-bf88-7368396eed5f	411	1982/01/01	12.7	3
bldg_238c47d3-ac15-4fdf-bfc5-830d08d6d20f	402	1983/01/01	28.8	8
bldg_0c870c4b-0362-4354-a79c-736470d932b2	414	1962/01/01	16.1	2
bldg_0bbb4bf2-086e-4741-a729-12a1c8e6717f	454	« NULL »	« NULL »	« NULL »
bldg_335fc27e-5931-4fcc-a05a-be9d279b42e3	411	1947/01/01	« NULL »	1
bldg_4b7ca424-4d77-4756-b1c3-383045febd5c	411	1961/01/01	5.8	2
bldg_9c97b283-c9e4-4dbd-abf0-81c72c0be7c8	431	1981/01/01	« NULL »	2
bldg_d8201b21-0bd9-4805-882e-e34541c981bb	411	1953/01/01	« NULL »	2

図 2-44 建物モデル属性の公開用 API

次に、PostgREST の設定を実施した。公式サイトに記載されているダウンロード先（https://github.com/PostgREST/postgrest/releases/latest）から PostgREST のバイナリをダウンロードする。ダウンロードしたファイルは圧縮されているため解凍し、/usr/local/bin/以下に解凍したデータ一式を配置する。その後、3DCityDB と接続するための設定ファイル（*.conf）を作成し、PostgREST の起動コマンドを実行すると PostgREST の動作が開始する。

【PostgREST の設定ファイル】

```
# db-uri = "postgres://user:pass@host:port/dbname"
db-uri = "postgres://authenticator:password@localhost:5432/citydb_v4"
db-schemas = "citydb"
db-anon-role = "web_anon"
```

【PostgREST の設定作業】

```
# 圧縮ファイルを解凍
tar xJf postgrest-v11.1.0-linux-static-x64.tar.xz
mv postgrest /usr/local/bin/
# 設定ファイルの作成
touch citydb_v4.conf
vi citydb_v4.conf
# postgrest の起動
postgrest citydb_v4.conf
```

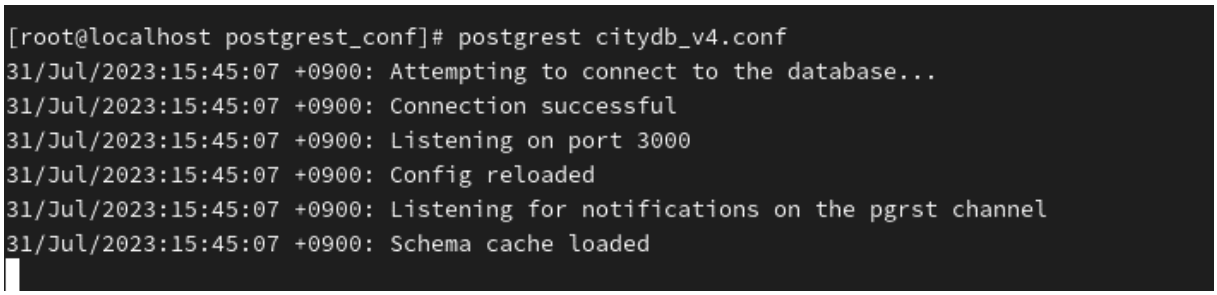


図 2-45 PostgREST の起動後画面

PostgREST が起動した後に、別のコンソール端末から以下コマンドを実行し、公開用の建物モデルの属性情報が取得できれば、PostgREST は問題なく動作している状態である。

【PostgREST の挙動確認】

```
curl http://localhost:3000/building_attr_view
```

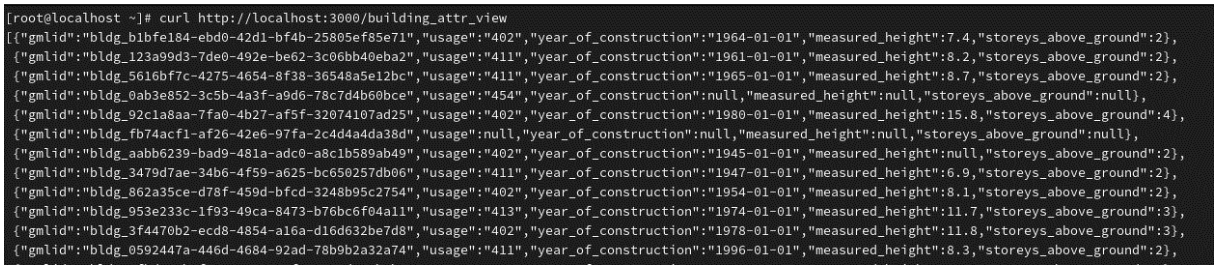


図 2-46 PostgREST の挙動確認結果

最後に、3DCityDB-Web-Map-Client 上での動作確認を実施した。3DCityDB-Web-Map-Client の Add / Configure Layer 機能から建物モデルを登録する際に、属性情報に関する以下の 2 つのパラメータを PostgREST 用の設定に変更する必要がある。

【Add / Configure Layer 機能の PostgREST 用設定】

- thematicDataUrl
  公開用 API を指定する
  http://localhost:3000/building_attr_view
- Thematiic Data Source
  PostgesSQL REST API を選択する

その他のパラメータは適当なものを入力し、Add layer ボタンを押下して建物モデルの登録を完了した後、メインビューに表示される建物モデルを選択するとウィンドウ右上に公開用 API で定義した属性情報が表示された。

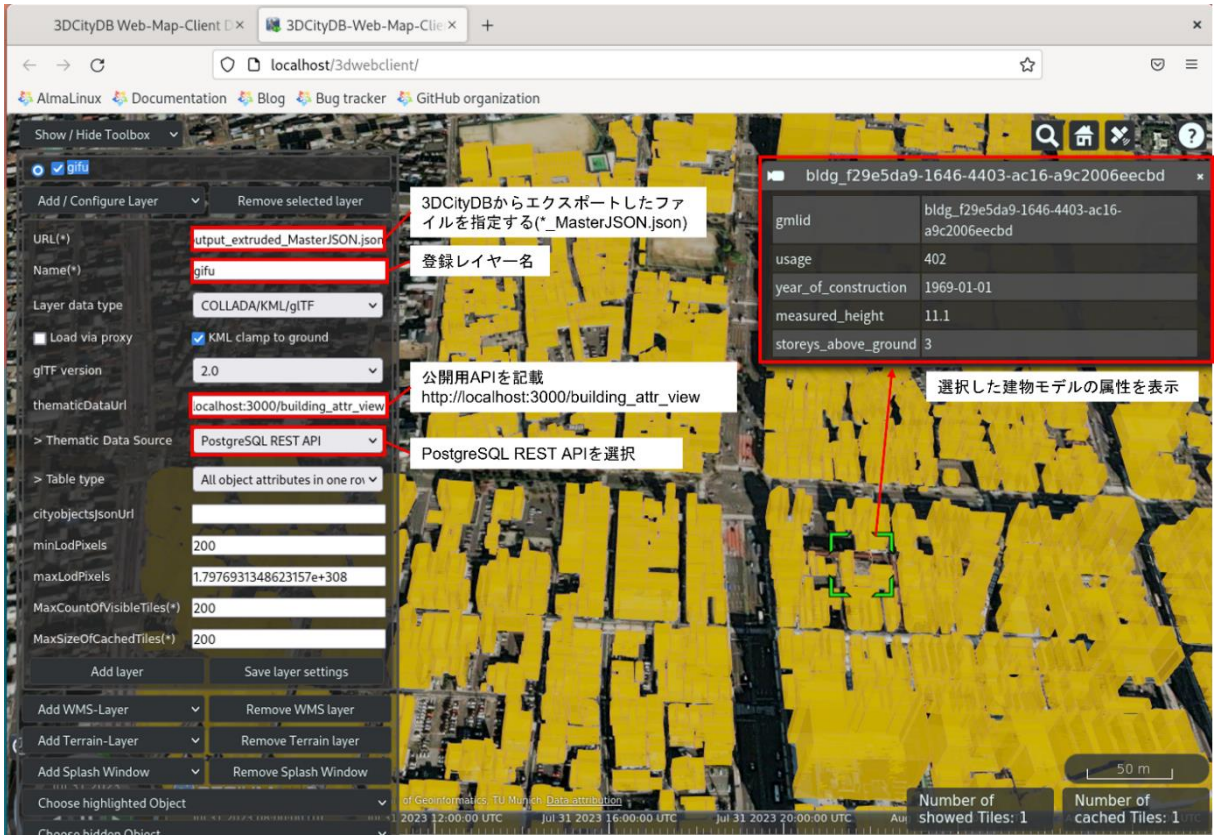


図 2-47 3DCityDB-Web-Map-Client と PostgREST の連携結果

RESTful API を利用した属性情報の参照は、3DCityDB に公開用の API を用意したり、RESTful API から 3DCityDB へアクセス可能なようにロールを追加したりと事前準備が必要となるが、一度設定を行えば、データベースのスキーマに変更が発生しない限り、データベースの属性値の更新が行われてもビューア側は RESTful API による属性取得のみで最新の値を得ること出来るという利点がある。一方、スプレッドシートによる属性値の取得をビューアが行っている場合は、属性値の更新に伴ってスプレッドシートの更新を行う手間が発生する。

公開用の API は、作成者が自由に作成できるため、地物ごとに属性を公開したり、地物同士を連携した属性を公開したりと、用途に合わせて公開する属性をカスタマイズすることが可能である。

2.7.3. PostgREST を使用した DB 更新方法の確認

CityGML データ内の特定の地物の情報を更新する場合、現状では地物が存在する図郭を探索するという一手間をかけてから、該当図郭の CityGML ファイル内に存在する地物の情報を更新する必要がある。また、CityGML ファイルを更新した後は PLATEAU VIEW で表示可能なように、CityGML データから 3DTiles データに変換する作業等も必要である。例えば、この特定地物の情報を更新する作業が、モデルビューア上から更新対象の地物を選択し、属性値を編集することで可能となるならば、更新作業が簡易になると考える。前項で記載したとおり、RESTful API を利用してビューア上から直接データベース上で管理されている地物の属性値を取得できることが確認できているため、本項では RESTful API を利用した地物の属性値の更新が可能であるか確認する。

3DCityDB に付随しているビューアである 3DCityDB-Web-Map-Client や PLATEAU VIEW でも CityGML データのモデル形状を表示するためにはデータ変換が必要であるため、 RESTful API による属性値の更新が可能であっても、モデル形状をビューアに変換するためのデータ変換作業は残ると考える。 PostgREST の公式ドキュメントのチュートリアル（https://postgrest.org/en/stable/tutorials/tut0.html）を参考に、PostgREST を利用して 3DCityDB の建物モデルの属性値を更新する方法の確認を実施した。前提として、3DCityDB の building テーブル内に建物モデル情報が格納されており、更新対象となる建物の ID が明確になっているものとする。

まず、3DCityDB に対して building テーブル更新用のロールを作成する。

【3DCityDB の変更作業】

```
# DB に接続
psql -h localhost -p 5432 -U postgres -d citydb_v4
# 更新用のロール作成
create role update_user nologin;
# authenticator に update_user と同等の権限を付与
grant update_user to authenticator;
# update_user に citydb スキーマのアクセスを許可
grant usage on schema citydb to update_user;
# update_user に建物テーブルのアクセス権限を付与
grant all on citydb.building to update_user;
```

次に、作成したロールに対して、認証用のアクセストークンを設定する。なお、アクセストークンを作成するために 32 桁以上のパスワードが必要となる。今回は、32 桁の乱数を生成してパスワードとして利用する。

生成したパスワードは、PostgREST の設定ファイル（citydb_v4.conf）に追記する。なお、設定ファイル更新後は、PostgREST を再起動する必要がある。

【パスワード例】

QbuPvc2JBGDzWGzeMuQjPYjcfePBMCpc

【PostgREST の設定ファイル】

```
# db-uri = "postgres://usr:pass@host:port/dbname"
db-uri = "postgres://authenticator:password@localhost:5432/citydb_v4"
db-schemas = "citydb"
db-anon-role = "web_anon"
jwt-secret = "QbuPvc2JBGDzWGzeMuQjPYjcfePBMCpc"#追記部分
```

次に、以下のサイトにおいて、Decoded 部分に必要事項を入力し、Encoded 部分に表示されるアクセストークンを取得する。

https://jwt.io/

【Decoded 部分の設定内容】

HEADER　　：　デフォルト値から変更なし

PAYLOAD　　：　{“role” ： “update_user”}

VERIFY SIGNATURE　　：　テキストボックス内に生成したパスワードを記載する

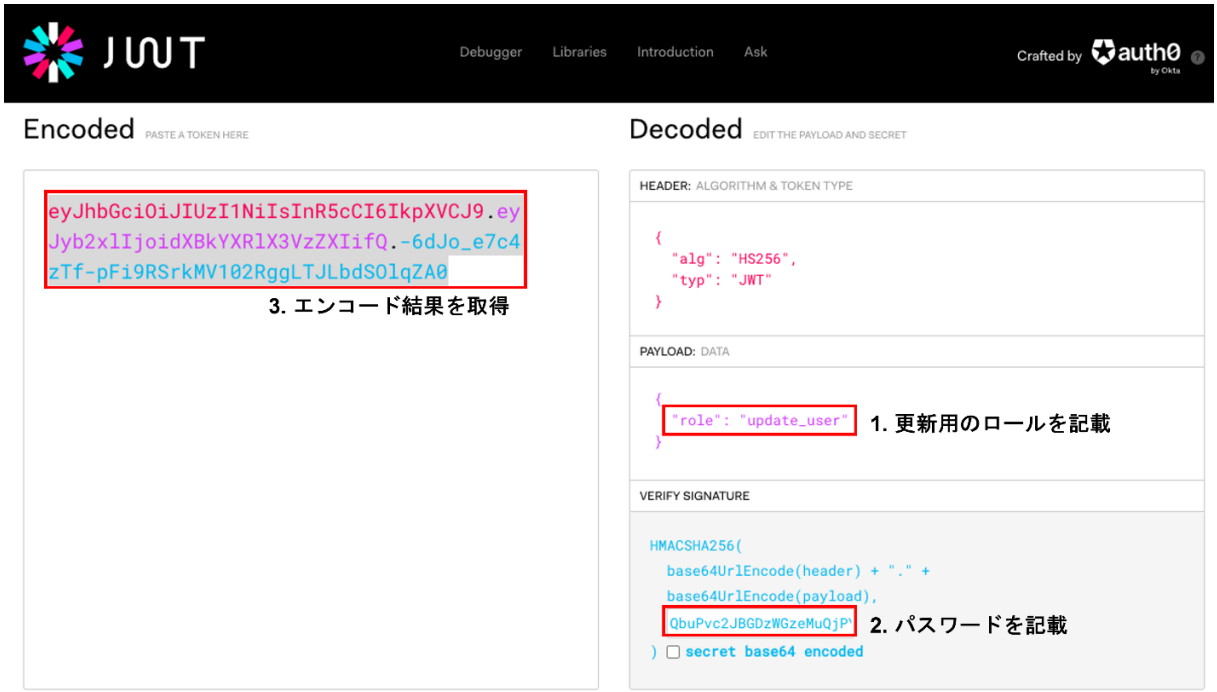


図 2-48 アクセストークンの作成

取得したアクセストークンを環境変数に登録する。

【アクセストークンの環境変数登録】

```
export TOKEN="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoidXBkYXRlX3VzZXIifQ.-6dJo_e7c4zTf-pFi9RSrkMV102RggLTJLbdSOlqZA0"
```

最後に、building テーブルのデータに対して更新作業を行い、動作を確認する。
更新対象のデータは id=34596 のデータとし、measured_height の値を更新する。

【building テーブルの更新作業の挙動確認】

```
export TOKEN="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoidXBkYXRlX3VzZXIifQ.-6dJo_e7c4zTf-pFi9RSrkMV102RggLTJLbdSOlqZA0"
```

```
curl http://localhost:3000/building?id=eq.34596 -X PATCH -H "Authorization: Bearer $TOKEN"
```

http://localhost:3000/building?id=eq.34596	:	ポート番号の後に更新対象のテーブル名（building）を記載し、?の後に更新対象のデータの条件（id=eq.34596）を記載する
-H "Authorization: Bearer $TOKEN"	:	認証用アクセストークンとして、環境変数 TOKEN に出力したアクセストークンを指定する
-d '{"measured_height": 10.0}'	:	変更値を JSON 形式で記載する

JSON

PostgREST による building テーブルの更新後に、building テーブルの id=34596 のデータを確認したところ、mesured_height が NULL から 10 に設定値が変更になっていることを確認した。

更新前

id	objectclass_id	building	buildi	class	class_c	function	function_c	usage	usage_codespace	year_of_constru	year_of_dem	roof_type	roof_type_codespace	measured_height	measured_
34521	26	« NULL	34521	« NULL	« NULL	« NULL »	« NULL »	411	../../codelists/Building_usage.xml	1946/01/01	« NULL »	« NULL »	« NULL »	10.2	m
34522	26	« NULL	34522	« NULL	« NULL	« NULL »	« NULL »	402	../../codelists/Building_usage.xml	1964/01/01	« NULL »	« NULL »	« NULL »	7.4	m
34524	26	« NULL	34524	« NULL	« NULL	« NULL »	« NULL »	411	../../codelists/Building_usage.xml	1946/01/01	« NULL »	« NULL »	« NULL »	7.5	m
34530	26	« NULL	34530	« NULL	« NULL	« NULL »	« NULL »	411	../../codelists/Building_usage.xml	1961/01/01	« NULL »	« NULL »	« NULL »	8.2	m
34559	26	« NULL	34559	« NULL	« NULL	« NULL »	« NULL »	415	../../codelists/Building_usage.xml	1975/01/01	« NULL »	« NULL »	« NULL »	9.4	m
34567	26	« NULL	34567	« NULL	« NULL	« NULL »	« NULL »	411	../../codelists/Building_usage.xml	1965/01/01	« NULL »	« NULL »	« NULL »	8.7	m
34568	26	« NULL	34568	« NULL	« NULL	« NULL »	« NULL »	415	../../codelists/Building_usage.xml	1991/01/01	« NULL »	« NULL »	« NULL »	8.5	m
34578	26	« NULL	34578	« NULL	« NULL	« NULL »	« NULL »	454	../../codelists/Building_usage.xml	« NULL »	« NULL »	« NULL »	« NULL »	« NULL »	« NULL »
34579	26	« NULL	34579	« NULL	« NULL	« NULL »	« NULL »	431	../../codelists/Building_usage.xml	« NULL »	« NULL »	« NULL »	« NULL »	24.7	m
34589	26	« NULL	34589	« NULL	« NULL	« NULL »	« NULL »	402	../../codelists/Building_usage.xml	1980/01/01	« NULL »	« NULL »	« NULL »	15.8	m
34596	26	« NULL	34596	« NULL	« NULL	« NULL »	« NULL »	« NULL	« NULL »	« NULL »	« NULL »	« NULL »	« NULL »	« NULL »	« NULL »
34609	26	« NULL	34609	« NULL	« NULL	« NULL »	« NULL »	402	../../codelists/Building_usage.xml	1945/01/01	« NULL »	« NULL »	« NULL »	« NULL »	« NULL »

更新後

id	objectclass_id	building	buildi	class	class_c	function	function_c	usage	usage_codespace	year_of_constru	year_of_dem	roof_type	roof_type_codespace	measured_height	measured_
34521	26	« NULL	34521	« NULL	« NULL	« NULL »	« NULL »	411	../../codelists/Building_usage.xml	1946/01/01	« NULL »	« NULL »	« NULL »	10.2	m
34522	26	« NULL	34522	« NULL	« NULL	« NULL »	« NULL »	402	../../codelists/Building_usage.xml	1964/01/01	« NULL »	« NULL »	« NULL »	7.4	m
34524	26	« NULL	34524	« NULL	« NULL	« NULL »	« NULL »	411	../../codelists/Building_usage.xml	1946/01/01	« NULL »	« NULL »	« NULL »	7.5	m
34530	26	« NULL	34530	« NULL	« NULL	« NULL »	« NULL »	411	../../codelists/Building_usage.xml	1961/01/01	« NULL »	« NULL »	« NULL »	8.2	m
34559	26	« NULL	34559	« NULL	« NULL	« NULL »	« NULL »	415	../../codelists/Building_usage.xml	1975/01/01	« NULL »	« NULL »	« NULL »	9.4	m
34567	26	« NULL	34567	« NULL	« NULL	« NULL »	« NULL »	411	../../codelists/Building_usage.xml	1965/01/01	« NULL »	« NULL »	« NULL »	8.7	m
34568	26	« NULL	34568	« NULL	« NULL	« NULL »	« NULL »	415	../../codelists/Building_usage.xml	1991/01/01	« NULL »	« NULL »	« NULL »	8.5	m
34578	26	« NULL	34578	« NULL	« NULL	« NULL »	« NULL »	454	../../codelists/Building_usage.xml	« NULL »	« NULL »	« NULL »	« NULL »	« NULL »	« NULL »
34579	26	« NULL	34579	« NULL	« NULL	« NULL »	« NULL »	431	../../codelists/Building_usage.xml	« NULL »	« NULL »	« NULL »	« NULL »	24.7	m
34589	26	« NULL	34589	« NULL	« NULL	« NULL »	« NULL »	402	../../codelists/Building_usage.xml	1980/01/01	« NULL »	« NULL »	« NULL »	15.8	m
34596	26	« NULL	34596	« NULL	« NULL	« NULL »	« NULL »	« NULL	« NULL »	« NULL »	« NULL »	« NULL »	« NULL »	10	« NULL »
34609	26	« NULL	34609	« NULL	« NULL	« NULL »	« NULL »	402	../../codelists/Building_usage.xml	1945/01/01	« NULL »	« NULL »	« NULL »	« NULL »	« NULL »

図 2-49 building テーブルの更新作業の結果

なお、3DCityDB 上において同様のことを行う場合は、以下のような SQL の UPDATE 文となる。

【3DCityDB での更新例】

```
# 3DCityDB に接続
psql -h localhost -p 5432 -U postgres -d citydb_v4
# 更新
UPDATE building SET measured_height=10 WHERE id=34596;
# 確認
SELECT id, measured_height FROM building WHERE id=34596;
  id   | measured_height
-------+-----------------
 34596 |              10
```

モデル形状の更新に関しては、3DCityDB 上において以下のような SQL の UPDATE 文を用いてモデル形状を更新できることを確認した。以下に示す例では建物モデルのある面の頂点の高さを UPDATE 文を用いて更新している。3DCityDB では、面の形状データを surface_geometry テーブルの geometry カラムで管理している。本検証では PostgreSQL と PostGIS を用いて 3DCityDB の環境を作成している関係上、geometry カラムのデータは PostGIS の geometry 型となる。この geometry 型は、PostGIS プログラムでのみ使用されるフォーマットであり可視性や利便性が良くないため、PostGIS に geometry 型の入出力のための関数が用意されている。モデル形状の更新確認では形状の座標値をテキスト形式に変換する ST_AsText()関数とテキスト形式の座標値を geometry 型のデータに変換する ST_GeomFromText()関数を使用した。

【3DCityDB でのモデル形状の更新例】

赤字部分が更新対象とした値である。

```
# 3DCityDB に接続
psql -h localhost -p 5432 -U postgres -d citydb_v4
# 現状の確認
SELECT id,ST_AsText(geometry) FROM surface_geometry WHERE id=36;
 id | st_astext
----+-----------------------------------------------------------------------
  36 | POLYGON Z ((139.796351967432 35.5330969485669 3.66410811,139.79634196185
3 35.5330921511014 3.66408749,139.79632067879 35.5331218372617 3.6635608,139.7
96330684084 35.5331266345334 3.66358141,139.796319965586 35.5331415846616 3.66
331703,139.796390136465 35.5331752211599 3.66346657,139.796400855826 35.533160
2799723 3.66373085,139.796408790681 35.5331640845147 3.66374829,139.7964300850
89 35.533134404299 3.66427499,139.796422149711 35.5331305994605 3.66425755,139
.796432657369 35.5331159533379 3.6645183,139.796362464501 35.5330823072676 3.6
6436872,139.796351967432 35.5330969485669 3.66410811))
# 更新
UPDATE surface_geometry SET geometry=ST_GeometryFromText('POLYGON((139.796351967432
35.5330969485669 7.66410811,139.796341961853 35.5330921511014 3.66408749,139.79632067879
35.5331218372617 3.6635608,139.796330684084 35.5331266345334 3.66358141,139.796319965586
35.5331415846616 3.66331703,139.796390136465 35.5331752211599 3.66346657,139.796400855826
35.5331602799723 3.66373085,139.796408790681 35.5331640845147 3.66374829,139.796430085089
35.533134404299 3.66427499,139.796422149711 35.5331305994605 3.66425755,139.796432657369
35.5331159533379 3.6645183,139.796362464501 35.5330823072676 3.66436872,139.796351967432
35.5330969485669 3.66410811))') WHERE id=36;
# 確認
SELECT id,ST_AsText(geometry) FROM surface_geometry WHERE id=36;
 id | st_astext
----+-----------------------------------------------------------------------
  36 | POLYGON Z ((139.796351967432 35.5330969485669 7.66410811,139.79634196185
3 35.5330921511014 3.66408749,139.79632067879 35.5331218372617 3.6635608,139.7
96330684084 35.5331266345334 3.66358141,139.796319965586 35.5331415846616 3.66
331703,139.796390136465 35.5331752211599 3.66346657,139.796400855826 35.533160
2799723 3.66373085,139.796408790681 35.5331640845147 3.66374829,139.7964300850
89 35.533134404299 3.66427499,139.796422149711 35.5331305994605 3.66425755,139
.796432657369 35.5331159533379 3.6645183,139.796362464501 35.5330823072676 3.6
6436872,139.796351967432 35.5330969485669 3.66410811))
```

本検討では未確認であるが、建物テーブル同様に更新用ロールに対して surface_geometry テーブルへのアクセス権を付与すれば RESTful API によるモデル形状の更新が行えると考える。ただし、現状では PLATEAU VIEW 等にモデル形状を表示するためにデータ変換作業が必要なため、更新後のモデルの形状を即時ビューアで確認するということは出来ない。

地物の属性や形状の更新が RESTful API 又は SQL 文にて行えることが確認できたため、RESTful API を利用した Web アプリによりデータベース更新や、SQL 文を利用したバッチ処理によるデータベース更新が行えると予想する。

2.8. i-UR 3.0 対応

3DCityDB 自体は CityGML 2.0 に準拠しており、i-UR には対応していない。PLATEAU が公開している 3D 都市モデル標準製品仕様書 ver.3.5

（https://www.mlit.go.jp/plateau/file/libraries/doc/plateau_doc_0001_ver03.pdf）では、i-UR 3.0（i-UR 1.4 に対して過年度の Project PLATEAU の検討成果を反映したもの）に対応している記載があるため、Project PLATEAU で作成した CityGML を 3DCityDB で管理するには、3DCityDB を i-UR 3.0 用に拡張する必要がある。

3DCityDB では、i-UR 1.4 拡張パッケージ（https://github.com/3dcitydb/iur-ade-citydb）を公開している。ADE Manager Plugin 適用済みの Importer/Exporter ツールで i-UR 1.4 拡張パッケージを読み込むと、3DCityDB 上で i-UR 1.4 のデータを管理することが可能となる。i-UR 1.4 拡張パッケージでは以下の処理を行っている。

【i-UR 1.4 拡張パッケージの処理】

- スキーマなどのデータベースの更新
- Importer/Exporter ツールの i-UR 1.4 データ対応

3DCityDB を i-UR 3.0 に対応させるには、最低限データベースの更新が必要であり、Importer/Exporter ツールを使用してデータのインポート/エクスポートを行う場合は Importer/Exporter ツールの i-UR 3.0 対応が必要となる。

ADE Manager Plugin 適用済みの Importer/Exporter ツールでは、ADE の XML スキーマからデータベースのスキーマを自動作成可能なため、i-UR 3.0 の XML スキーマを基にデータベースの更新作業を自動で行うことが可能である。

i-UR 2.0 へのデータベース拡張は、 ADE Manager を用いることで適用できることを確認した。Import/Export 機能の i-UR 2.0 対応は、プラグイン開発（Java）が必要である。

2.8.1. コード解析

Import/Export 機能の i-UR 2.0 対応のために、Java コードの解析を行った。

表 2-6 コードの構成

パス/ファイル名	処理内容
src¥main¥java¥org¥citydb¥ade¥iur¥schema	-
ADETable.java	テーブル名一覧を定義
ADETableMapper.java	テーブル内のカラム定義
src¥main¥java¥org¥citydb¥ade¥iur¥importer	-
¥urf、¥urg、¥uro、¥urt	テーブルごとにファイルを分けインポート処理を実装
ImportManager.java	テーブルのインポート処理を呼出し
src¥main¥java¥org¥citydb¥ade¥iur¥exporter	-
¥urf、¥urg、¥uro、¥urt	テーブルごとにファイルを分けエクスポート処理を実装
ExportManager.java	各テーブルのエクスポート処理を呼出し

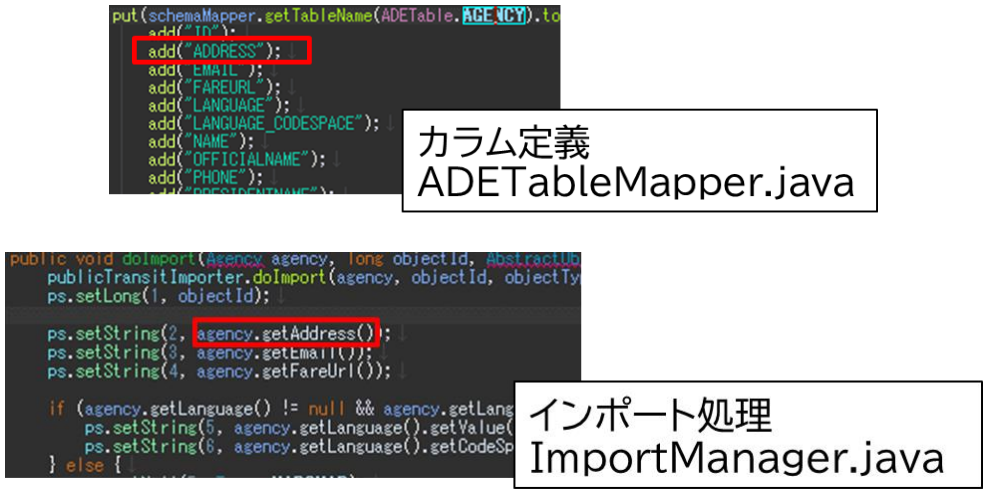


図 2-50 コード上のカラム定義とインポート処理の例

(1) lur-abe-citydb

■Importer クラス

Src/main/java/org/citydb/ade/iur/importer に urf、urg、uro、urt と各種類に応じたテーブルごと（属性ごと）の importer クラスが作成されている。

・i-UR 1.4 の場合

コンストラクタに DB へデータを挿入するための SQL ステートメントを宣言する。

doImport 関数にコンストラクタで宣言したステートメントに値を設定しバッチ処理を実行している。

```
public class BuildingPropertiesImporter implements UrbanObjectModuleImporter {
    private final CityGMLImportHelper helper;
    private final SchemaMapper schemaMapper;
    private final PreparedStatement ps;

    private final BuildingDetailsImporter buildingDetailsImporter;
    private final LargeCustomerFacilitiesImporter largeCustomerFacilitiesImporter;
    private final KeyValuePairImporter keyValuePairImporter;

    private int batchCounter;

    public BuildingPropertiesImporter(Connection connection, CityGMLImportHelper helper, ImportManager manager) throws CityGMLImportException, SQLException {
        this.helper = helper;
        this.schemaMapper = manager.getSchemaMapper();

        ps = connection.prepareStatement("insert into " +
                helper.getTableNameWithSchema(schemaMapper.getTableName(ADETable.BUILDING)) + " " +
                "(id, buildingdetails_id, largecustomerfacilities_id) " +
                "values (?, ?, ?)");

        buildingDetailsImporter = manager.getImporter(BuildingDetailsImporter.class);
        largeCustomerFacilitiesImporter = manager.getImporter(LargeCustomerFacilitiesImporter.class);
        keyValuePairImporter = manager.getImporter(KeyValuePairImporter.class);
    }

    public void doImport(ADEPropertyCollection properties, AbstractBuilding parent, long parentId, FeatureType parentType) throws CityGMLImportException, SQLException {
        ps.setLong(1, parentId);

        long buildingDetailsId = 0;
        if (properties.contains(BuildingDetailsPropertyElement.class)) {
            BuildingDetailsPropertyElement property = properties.getFirst(BuildingDetailsPropertyElement.class);
            if (property.isSetValue() && property.getValue().isSetObject()) {
                buildingDetailsId = buildingDetailsImporter.doImport(property.getValue().getObject());
                property.setValue(null);
            }
        }

        if (buildingDetailsId != 0)
            ps.setLong(2, buildingDetailsId);
        else
            ps.setNull(2, Types.NULL);

        long largeCustomerFacilitiesId = 0;
        if (properties.contains(LargeCustomerFacilitiesPropertyElement.class)) {
            LargeCustomerFacilitiesPropertyElement property = properties.getFirst(LargeCustomerFacilitiesPropertyElement.class);
            if (property.isSetValue() && property.getValue().isSetObject()) {
                largeCustomerFacilitiesId = largeCustomerFacilitiesImporter.doImport(property.getValue().getObject());
                property.setValue(null);
            }
        }

        if (largeCustomerFacilitiesId != 0)
            ps.setLong(3, largeCustomerFacilitiesId);
        else
            ps.setNull(3, Types.NULL);

        ps.addBatch();
        if (++batchCounter == helper.getDatabaseAdapter().getMaxBatchSize())
            helper.executeBatch(schemaMapper.getTableName(ADETable.BUILDING));

        if (properties.contains(ExtendedAttributeProperty.class)) {
            for (ExtendedAttributeProperty property : properties.getAll(ExtendedAttributeProperty.class)) {
                if (property.isSetValue() && property.getValue().isSetObject())
                    keyValuePairImporter.doImport(property.getValue().getObject(), parentId);
            }
        }
    }
```

図 2-51 Importer クラス（i-UR 1.4）

・i-UR 2.0 対応

新たに i-UR 2.0 で増えたテーブルのクラスを作成する。

コンストラクタにテーブルへデータを挿入するための SQL ステートメントを宣言する。

doImport 関数に各プロパティに対応する値を設定しバッチ処理を実行する。

```
public BuildingIDAttributeImporter(Connection connection, CityGMLImportHelper helper, ImportManager manager) throws CityGMLImportException, SQLException {

    LogLevel logLevel = LogLevel.INFO;
    Logger log = Logger.getInstance();
    log.setConsoleLogLevel(logLevel);
    log.info("BuildingIDAttributeImporter doImport");

    this.helper = helper;
    this.schemaMapper = manager.getSchemaMapper();

    ps = connection.prepareStatement("insert into " +
            helper.getTableNameWithSchema(schemaMapper.getTableName(ADETable.UR_BUILDINGIDATTRIBUTE)) + " " +
            "(id, building_buildingidattrib_id, branchID, partID, prefecture, prefecture_codespace, city, city_codespace) " +
            "values (?, ?, ?, ?, ?, ?, ?, ?, ?)");

}

public long doImport(BuildingIDAttribute buildingIDAttribute) throws CityGMLImportException, SQLException {

    LogLevel logLevel = LogLevel.INFO;
    Logger log = Logger.getInstance();
    log.setConsoleLogLevel(logLevel);
    log.info("BuildingIDAttributeImporter doImport");

    long objectId = helper.getNextSequenceValue(schemaMapper.getSequenceName(ADESequence.BUILDINGIDATTRIBUTE_SEQ));

    ps.setLong(1, objectId);

    ps.setString(2, buildingIDAttribute.getBuildingID());

    if (buildingIDAttribute.getBranchID() != null)
        ps.setInt(3, buildingIDAttribute.getBranchID());
    else
        ps.setNull(3, Types.INTEGER);

    if (buildingIDAttribute.getPartID() != null)
        ps.setInt(4, buildingIDAttribute.getPartID());
    else
        ps.setNull(4, Types.INTEGER);

    if (buildingIDAttribute.getPrefecture() != null && buildingIDAttribute.getPrefecture().isSetValue()) {
        ps.setString(5, buildingIDAttribute.getPrefecture().getValue());
        ps.setString(6, buildingIDAttribute.getPrefecture().getCodeSpace());
    } else {
        ps.setNull(5, Types.VARCHAR);
        ps.setNull(6, Types.VARCHAR);
    }

    if (buildingIDAttribute.getCity() != null && buildingIDAttribute.getCity().isSetValue()) {
        ps.setString(7, buildingIDAttribute.getCity().getValue());
        ps.setString(8, buildingIDAttribute.getCity().getCodeSpace());
    } else {
        ps.setNull(7, Types.VARCHAR);
        ps.setNull(8, Types.VARCHAR);
    }

    ps.addBatch();
    if (++batchCounter == helper.getDatabaseAdapter().getMaxBatchSize())
        helper.executeBatch(schemaMapper.getTableName(ADETable.UR_BUILDINGIDATTRIBUTE));

    return objectId;
}
```

図 2-52 Importer クラス（i-UR 2.0）

■ImportManager クラス

Src/main/java/org/citydb/ade/iur/importer に作成されている。

・i-UR 1.4 の場合

ImportObject 関数 or importGenericApplicationProperties 関数によって Import クラスの doImport 関数を呼び出している。

・i-UR 2.0 対応

新規作成した Import クラスの doImport 関数の呼出しを追加する。

■ADESequence（列挙型）

Src/main/java/org/citydb/ade/iur/schema に作成されている。

・i-UR 1.4 の場合

属性ごとのシーケンス名を宣言する。

・i-UR 2.0 対応

i-UR 2.0 で追加されるシーケンス名を追加する。

■ADETable（列挙型）

Src/main/java/org/citydb/ade/iur/schema に作成されている。

・i-UR 1.4 の場合

属性ごとのテーブル名と使用する Importer を宣言する。

・i-UR 2.0 対応

i-UR 2.0 で追加されるテーブル名と Importer を追加する。

(2) Iur-ade-citygml4j-1.4.2

■属性名クラス

Src/main/java/org/citygml4j/ade/iur/model/ 以下に urf、urg、uro、urt と各種類に応じたテーブルごと（属性ごと）のクラスが作成されている。

・i-UR 1.4 の場合

属性ごとのクラスのメンバに XML で使用されているプロパティとプロパティごとの Set、Get 関数が宣言されている。

・i-UR 2.0 対応

i-UR 2.0 で追加される属性の Type クラスを作成し、メンバに XML で使用されているプロパティとプロパティごとの Set、Get 関数が宣言している。

■属性名 Property クラス

属性名クラスのプロパティクラス。Src/main/java/org/citygml4j/ade/iur/model/ 以下に urf,urg,uro,urt と各種類に応じた属性ごとにクラスが作成されている。

・i-UR 1.4 の場合

属性ごとにクラスが作成されている。

・i-UR 2.0 対応

i-UR 2.0 で追加された属性ごとにクラスを実装する。

■属性名 PropertyElement クラス

属性名クラスのプロパティクラス。Src/main/java/org/citygml4j/ade/iur/model/ 以下に urf,urg,uro,urt と各種類に応じたテーブルごと（属性ごと）のクラスが作成されている。

・i-UR 1.4 の場合

属性ごとにクラスが作成されている。

・i-UR 2.0 対応

i-UR 2.0 で追加された属性ごとにクラスを実装する。

■UrbanObjectMarshaller

・i-UR 1.4 の場合

Marshall 属性名関数、marshall 属性名 Property 関数、create 属性名関数、create 属性名 Property 関数が存在する。

・i-UR 2.0 対応

i-UR 2.0 で追加された属性で Marshall 属性名関数、marshall 属性名 Property 関数、create 属性名関数、create 属性名 Property 関数を実装する。

■UrbanObjectUnmarshaller

・i-UR 1.4 の場合

Unmarshal 関数に各属性の Property クラスを返す処理と、Unmarshal 属性名関数、Unmarshal 関数名 Property 関数が存在する。

・i-UR 2.0 対応

i-UR 2.0 で追加された属性の Property クラスを返す処理を Unmarshal 関数に追加。追加する属性の Unmarshal 属性名関数、Unmarshal 属性名 Property 関数を実装する。

■属性名 PropertyType クラス

Src-gen/main/java/jp/go/kantei/iur/_1_4 に各 urf,urg,uro,urt のテーブルごとに（属性ごとに）宣言されている。

・i-UR 1.4 の場合

属性ごとのクラスのメンバに Type クラスを宣言し、メンバの Type クラスへの Set,Get,isSet 関数が宣言されている。

・i-UR 2.0 対応

i-UR 2.0 で追加された属性の PropertyType クラスを作成し、メンバの Type クラスへの Set,Get,isSet 関数を宣言している。

■属性名 Type クラス

Src-gen/main/java/jp/go/kantei/iur/_1_4 に各 urf,urg,uro,urt の属性ごとに宣言されている。

・i-UR 1.4 の場合

属性ごとのクラスのメンバに XML で使用されているプロパティとプロパティごとの Set、Get 関数を宣言している。

・i-UR 2.0 対応

i-UR 2.0 で追加される属性の Type クラスを作成し、メンバに XML で使用されているプロパティとプロパティごとの Set、Get 関数を宣言している。

2.9. CityGML 3.0 対応

現状、Project PLATEAU と 3DCityDB は CityGML 2.0 に準拠しており、i-UR 3.0 データを考慮しなければ Project PLATEAU で作成した CityGML データを 3DCityDB で管理可能である。

3DCityDB では、次バージョン(ver.5)において CityGML 3.0 に対応する予定であるため、Project PLATEAU でサポートする CityGML のバージョンを 3.0 に変更する場合は、3DCityDB ver.5 を利用すると良いと考える。ただし、3DCityDB ver.5.0 のリリース時期については未定である。(https://3dcitydb-docs.readthedocs.io/en/latest/overview/history.html 参照)

3. 3DCityDB サーバーの利用手順書

CityGML 2.0 をデータベース化して格納、編集、品質検査、Web GIS ビューアなどのフロントエンドへのデータ配信を可能とする技術調査の事前調査として環境構築を行った 3DCityDB サーバーの利用手順を記載する。なお、本書では、Windows PC をクライアント PC として利用する。

3.1. サーバー環境

サーバー環境に関して以下に記載する。

OS	：	AlmaLinux 9.2
3DCityDB	：	
DB	：	PostgreSQL ver.15.3 PostGIS ver.3.3.3 3DCityDB ver.4.4 （座標系は EPSG:6697（JGD2011 + JGD2011 （vertical） height））
Importer/Exporter	：	ver.5.3.0
3DCityDB-Web-Map-Client	：	ver.1.9.1
PostgREST	：	ver.11.2.0

サーバーが提供する機能については以下のとおり。

表 3-1 サーバー機能

機能	概要
3DCityDB サーバー	CityGML 2.0 フォーマットの CityGML データを管理するための DB サーバー機能。
3DCityDB 出力モデルの閲覧機能（ 3DCityDB-Web-Map-Client ）	3DCityDB がエクスポートする 3D 都市モデルを閲覧するための ビューア機能。
3DCityDB の属性情報の配信	PostgREST を利用した 3DCityDB の属性情報を配信。（ 3DCityDB-Web-Map-Client において、属性情報表示するために利用） 構築したサーバーでは、建物モデルの一部の属性情報を PostgREST によって公開している。

3.2. クライアント環境

クライアント環境について以下に記載する.

OS	:	Windows 10 Pro
3DCityDB	:	
Importer/Exporter	:	ver.5.3.0
Web ブラウザ	:	Google Chrome （検証時：ver.116.0.5845.141） WebGL をサポートしている Web ブラウザを使用すること。
SQL クライアントアプリ	:	A5:SQL Mk-2 ver.2.18.3

3.2.1. Importer/Exporter のインストール方法

Importer/Exporter ツールのインストール方法について以下に示す。

【Importer/Exporter ツールのインストール方法】

1. Java のインストール

検証では、JDK ver.17.0.8 をインストールした。

以下の URL から、JDK17 のインストーラーを入手する。

https://www.oracle.com/java/technologies/downloads/#jdk17-windows

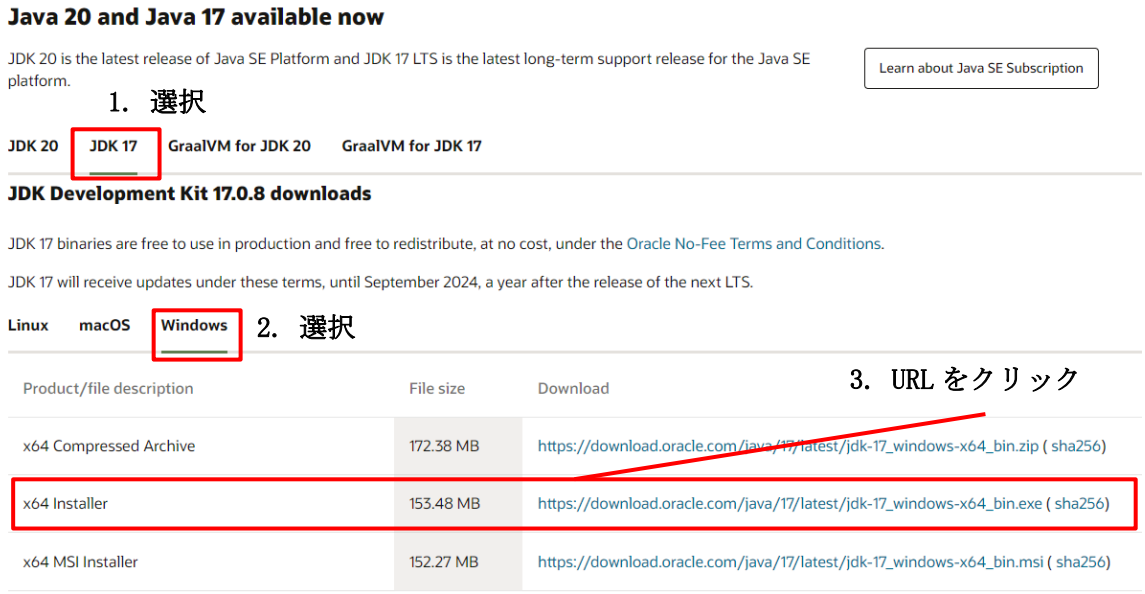


図 3-1 JDK インストーラーの入手

入手した jdk-17_windows-x64_bin.exe を起動し、下図を参考にインストールを行う。

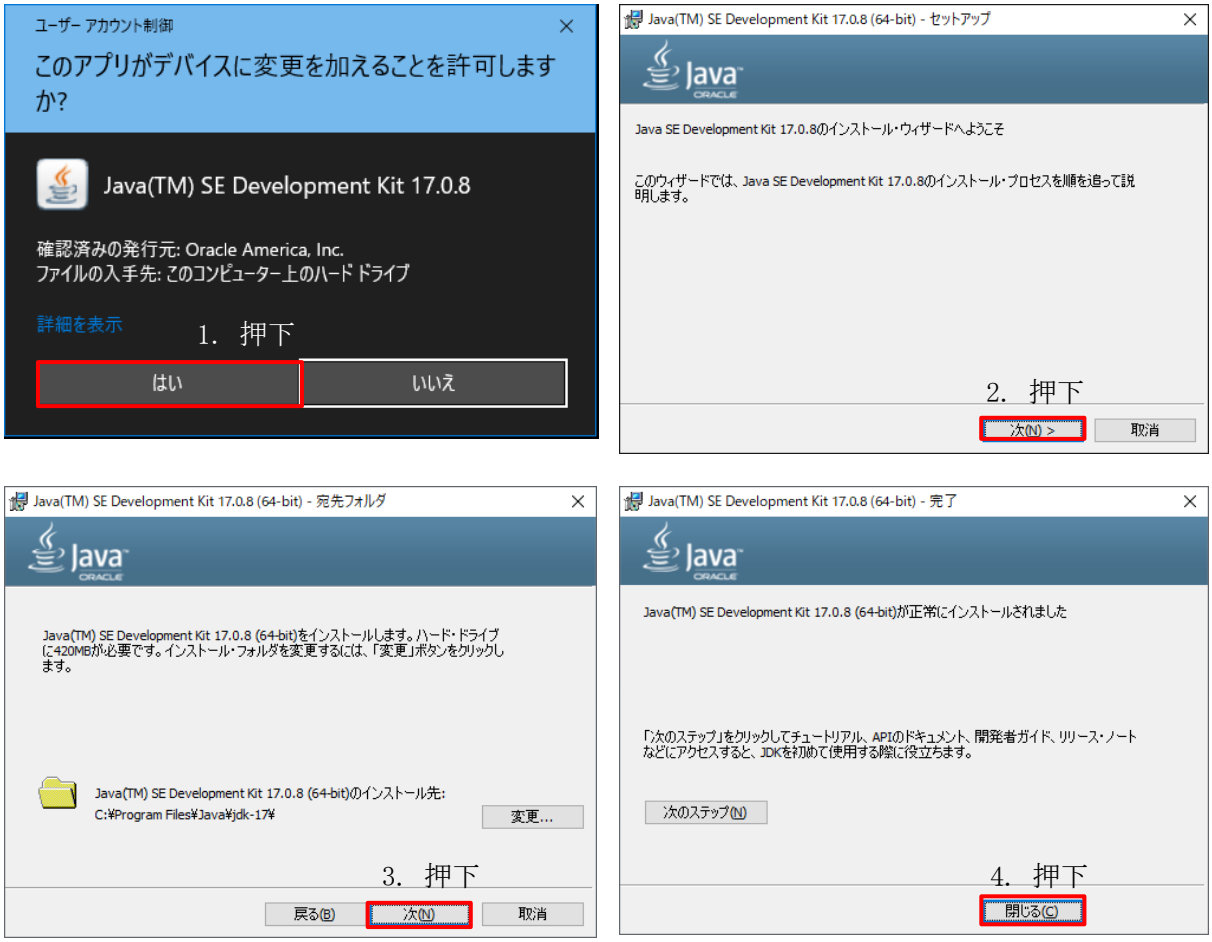


図 3-2 JDK のインストール

2. Importer/Exporter ツールのインストール

以下の URL から、インストーラーを入手する。

https://github.com/3dcitydb/3dcitydb-suite/releases

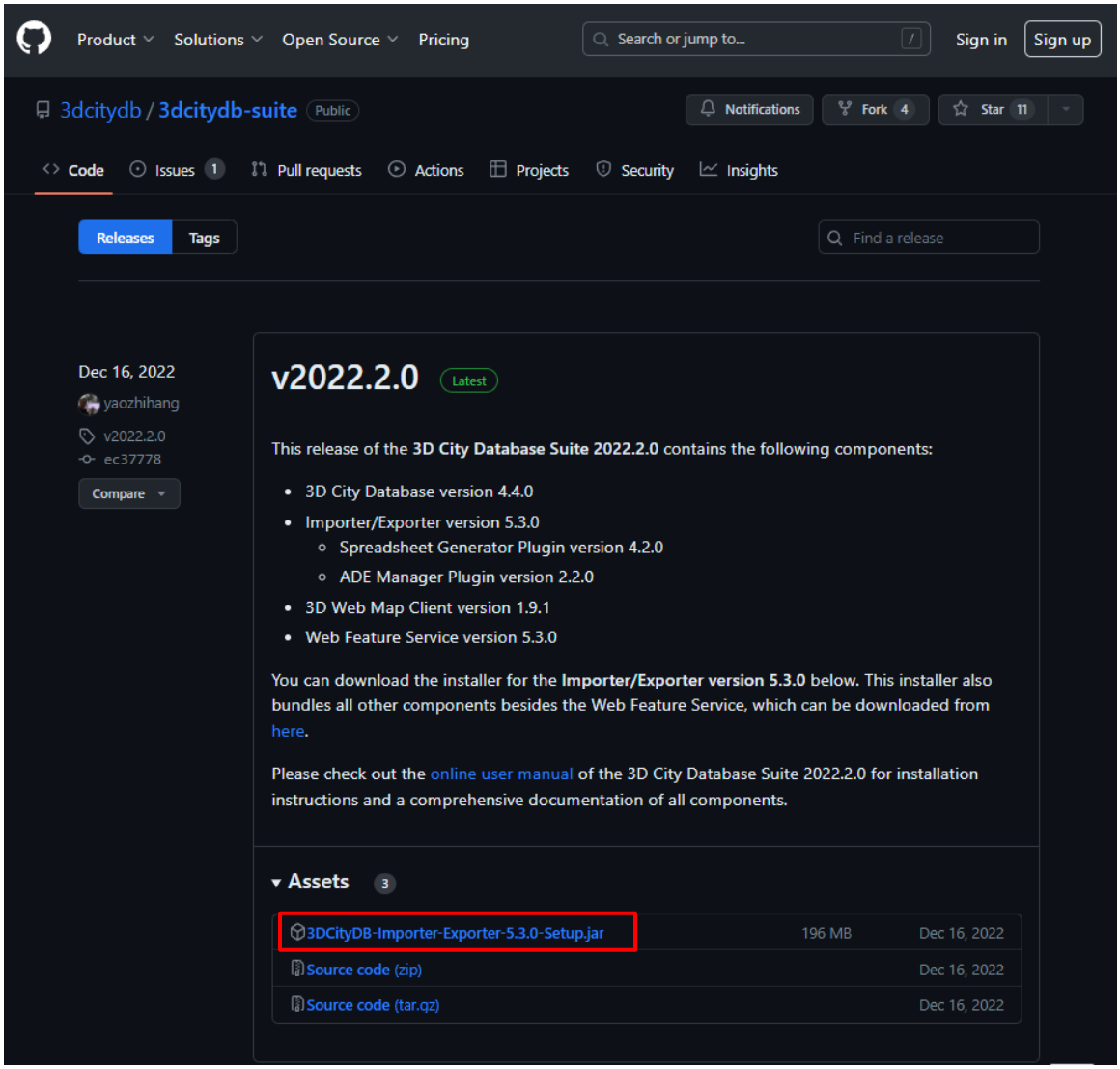


図 3-3 Importer/Exporter ツールのインストーラーの入手

入手した 3DCityDB-Importer-Exporter-5.3.0-Setup.jar をダブルクリックして、インストーラーを起動する。下図を参考にインストールを行う。

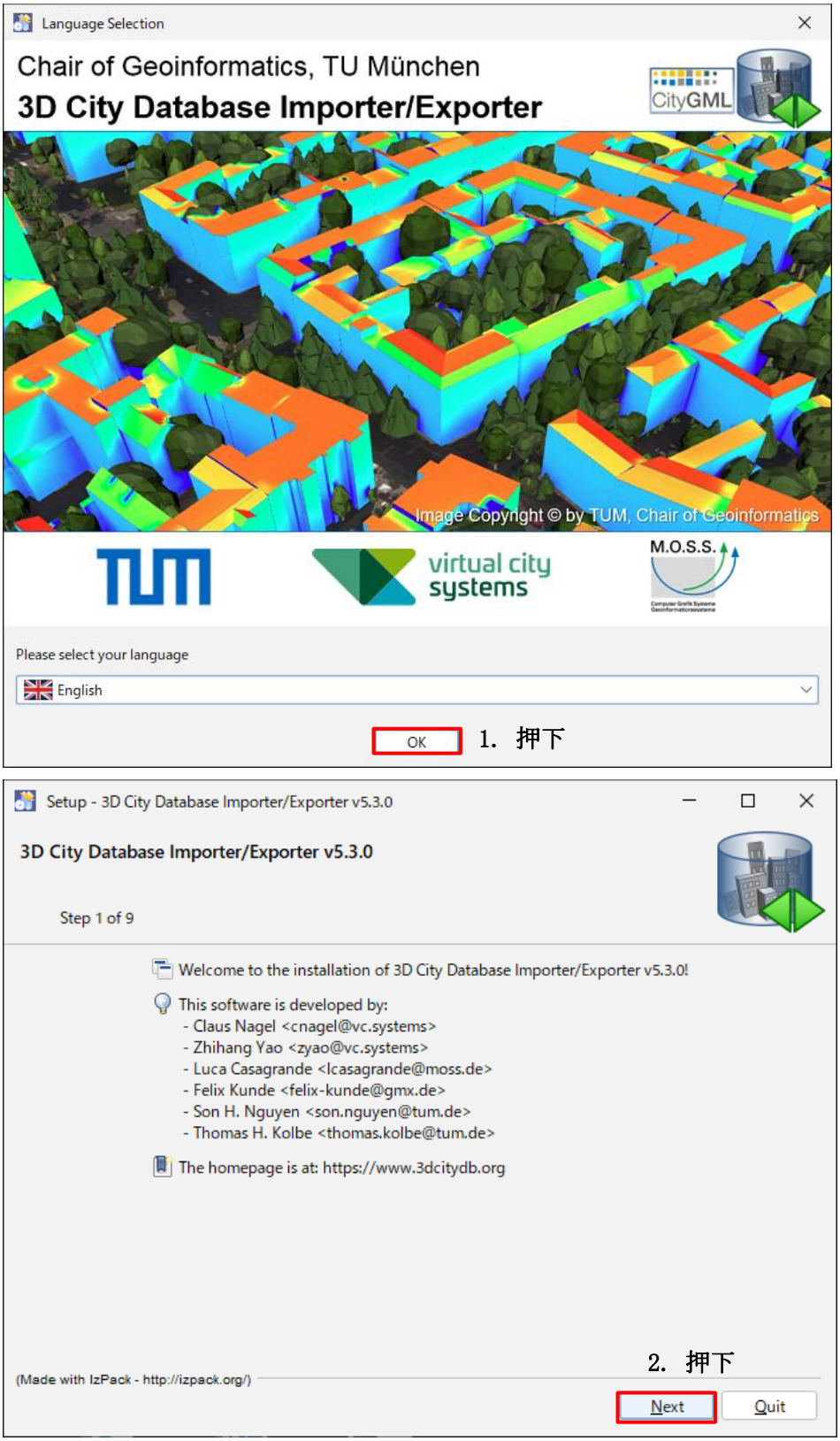
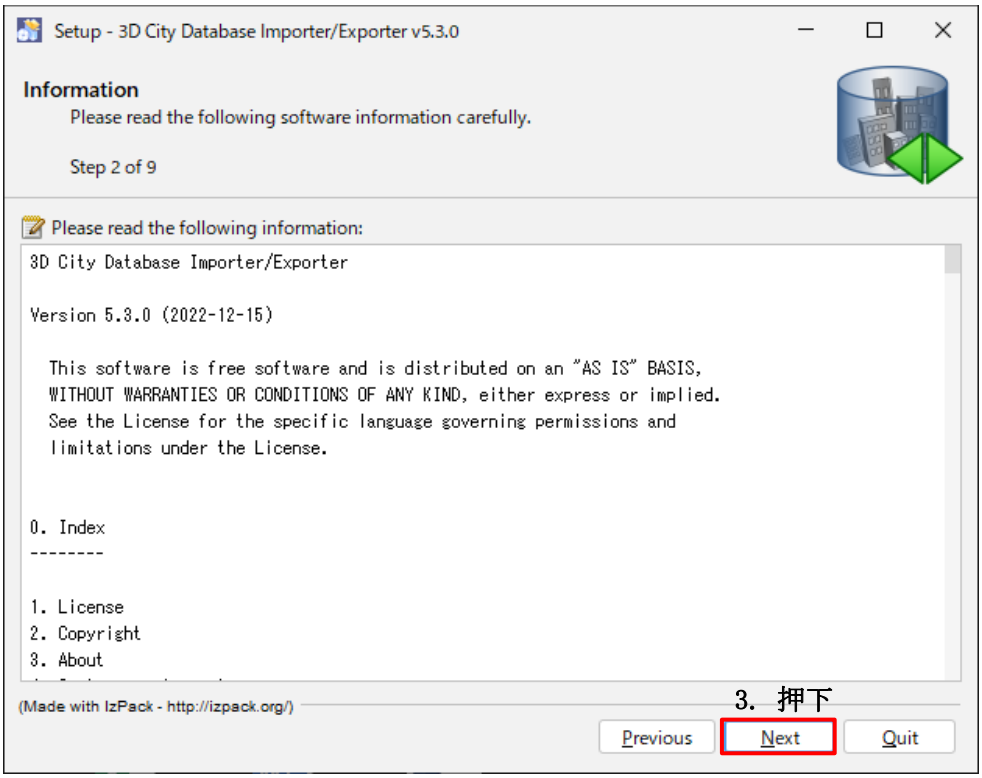


図 3-4 Importer/Exporter のインストール (1/5)



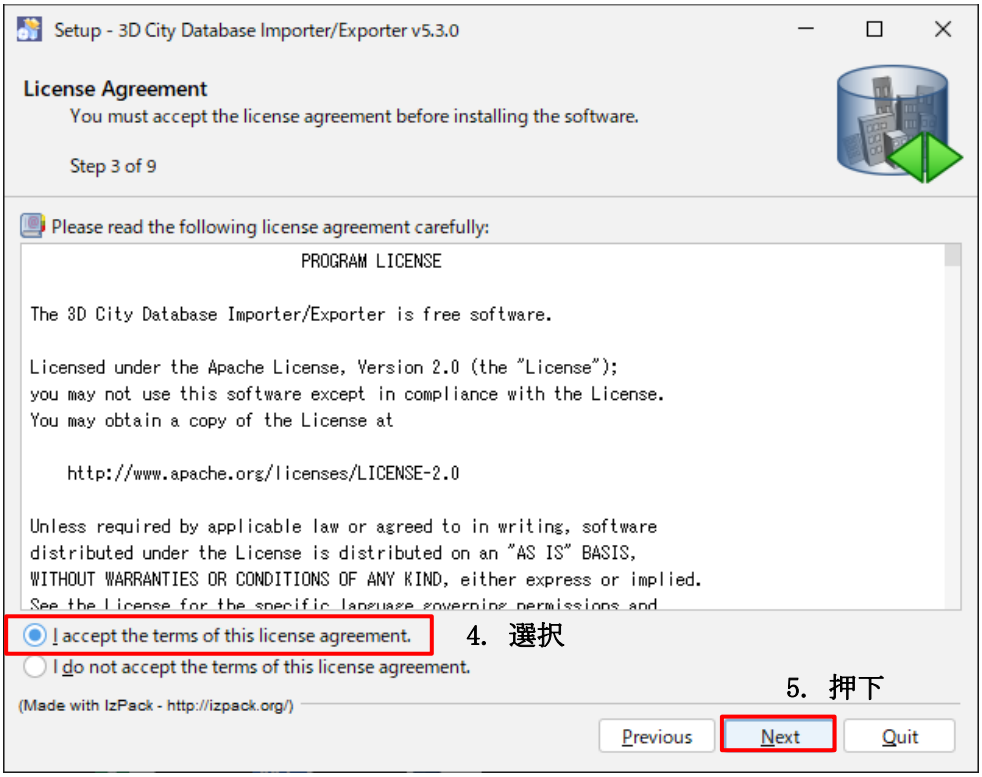


図 3-5 Importer/Exporter のインストール (2/5)

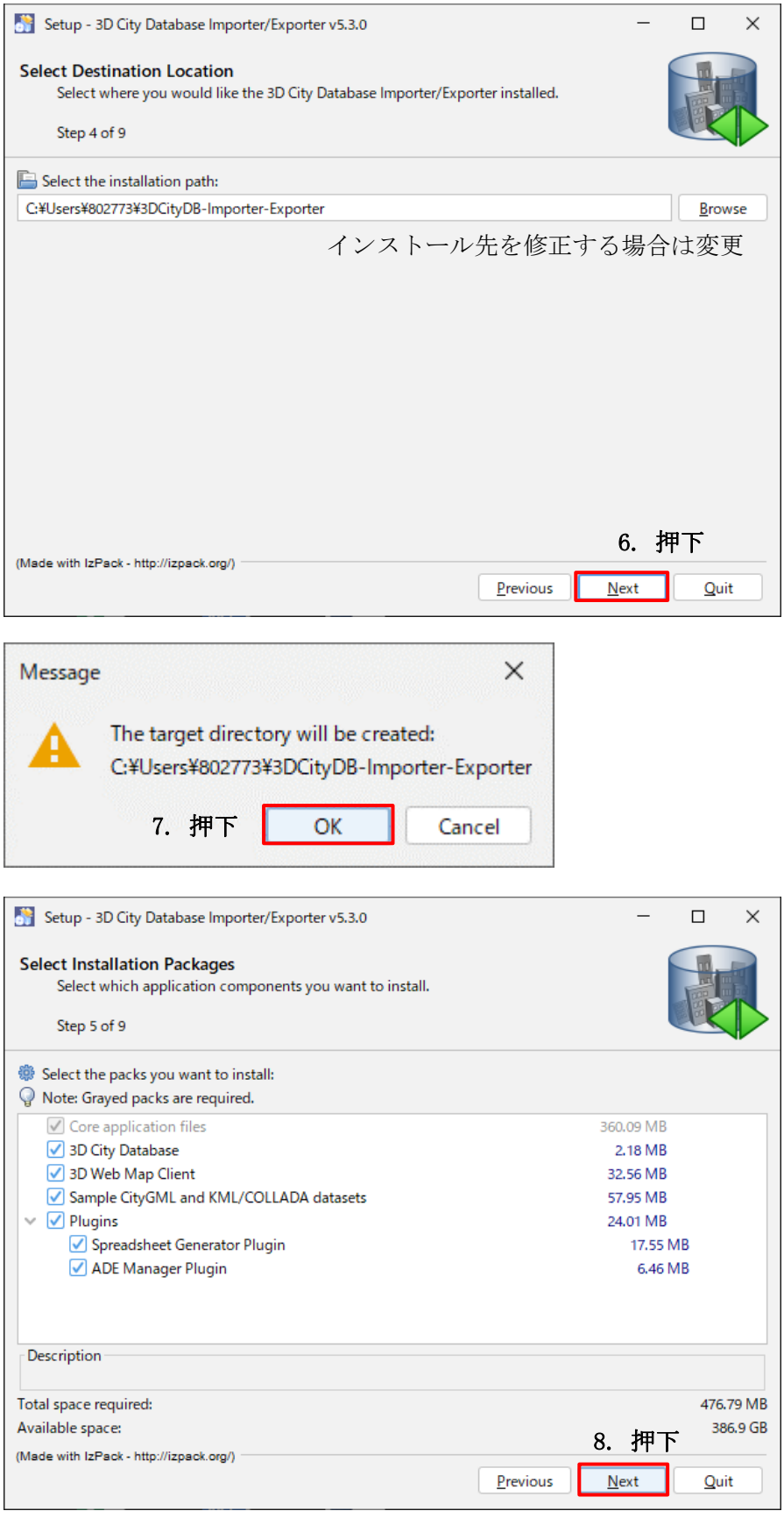
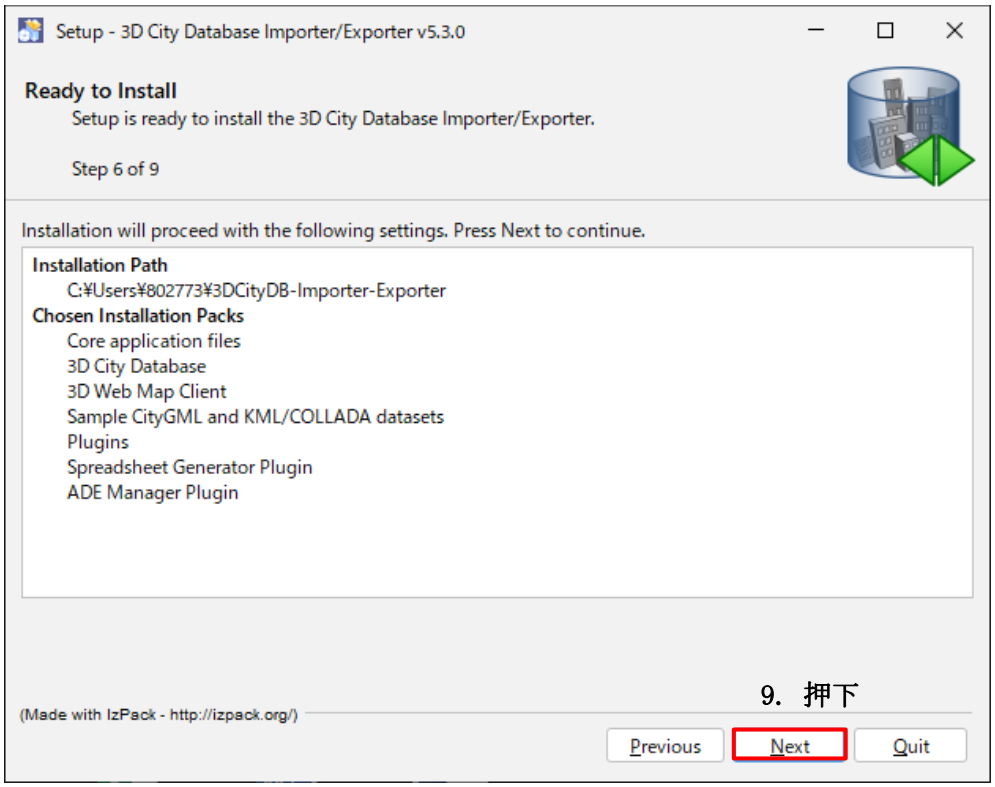


図 3-6 Importer/Exporter のインストール (3/5)



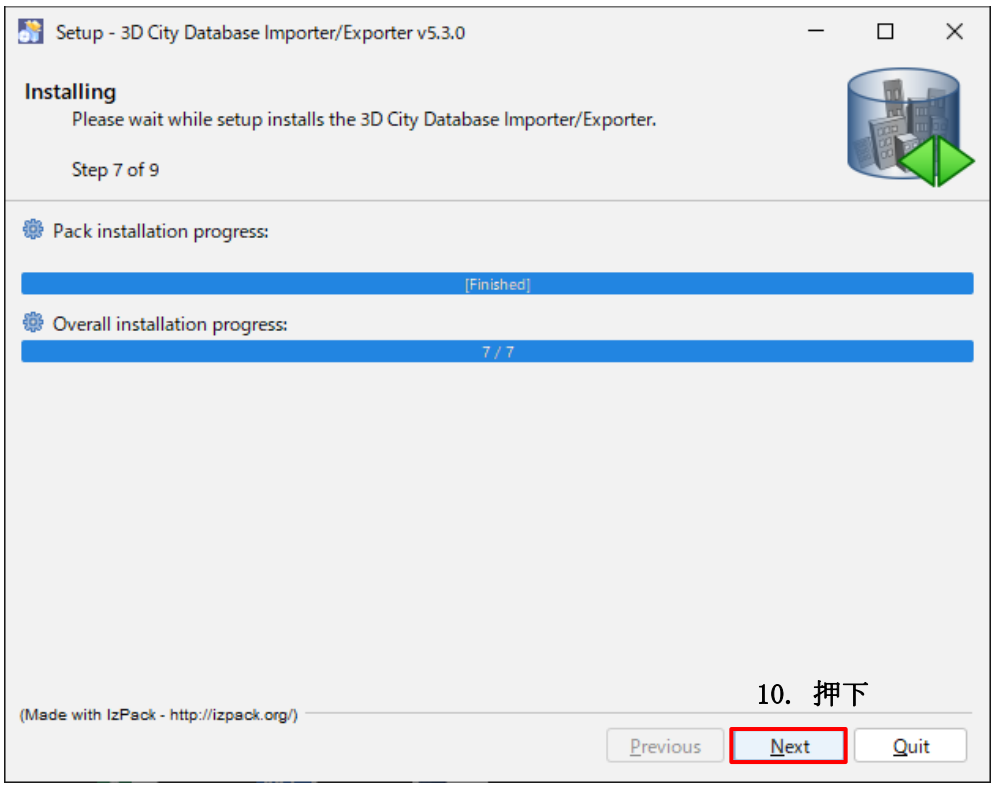


図 3-7 Importer/Exporter のインストール (4/5)

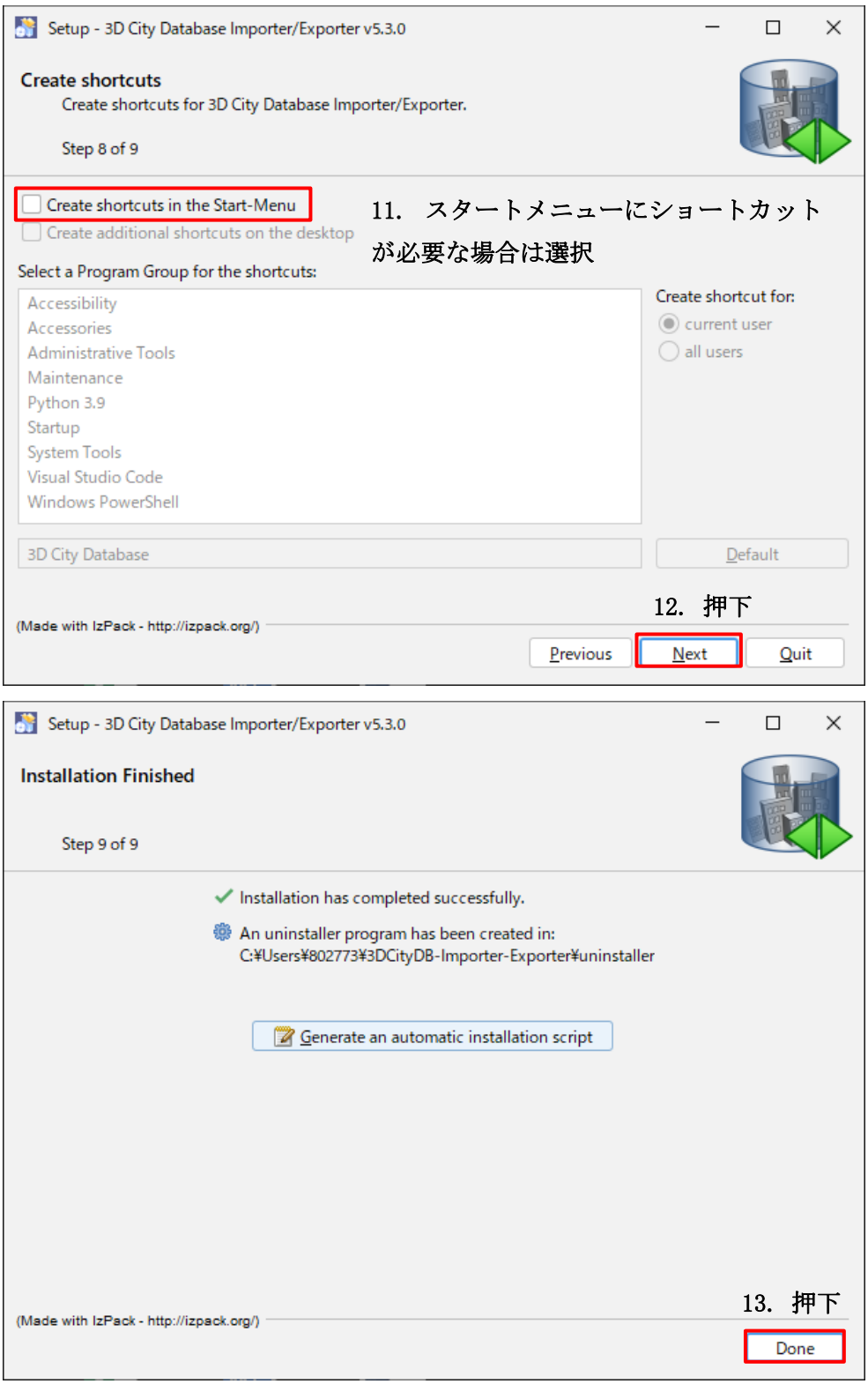


図 3-8 Importer/Exporter のインストール (5/5)

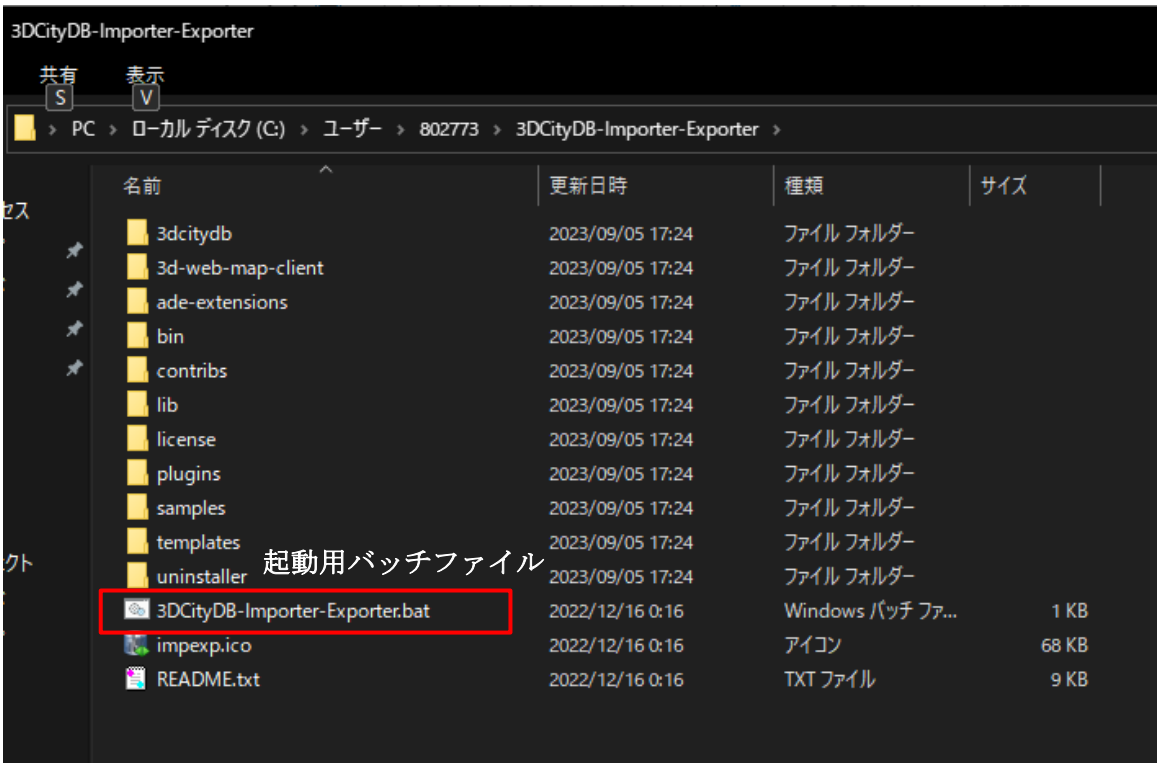


図 3-9 Importer/Exporter ツールインストール結果

3.2.2. A5:SQL Mk-2 のインストール方法

A5:SQL Mk-2 ツールのインストール方法について以下に示す。

【A5:SQL Mk-2 ツールのインストール方法】

1. 実行ファイル一式の入手

以下の URL からより、実行ファイル一式（ZIP ファイル）を入手する。

https://a5m2.mmatsubara.com/

図 3-10 A5:SQL Mk-2 の実行ファイル一式の入手

2. 取得した ZIP ファイルを解凍する

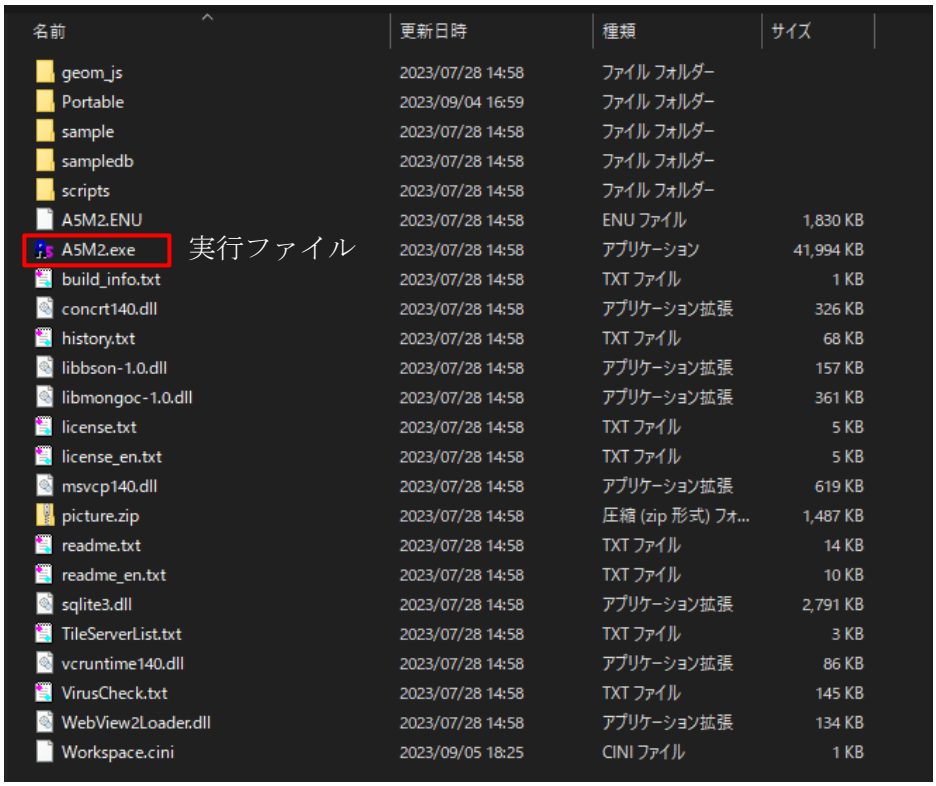


図 3-11 A5:SQL Mk-2 の解凍結果

3.3. DB 接続方法

3.3.1. Importer/Exporter ツールでの DB 接続

Importer/Exporter ツールにおける DB 接続方法を以下に示す。

【Importer/Exporter ツールにおける DB 接続方法】

1. Importer/Exporter ツールの起動

インストールフォルダ内に存在する 3DCityDB-Importer-Exporter.bat をダブルクリックする。

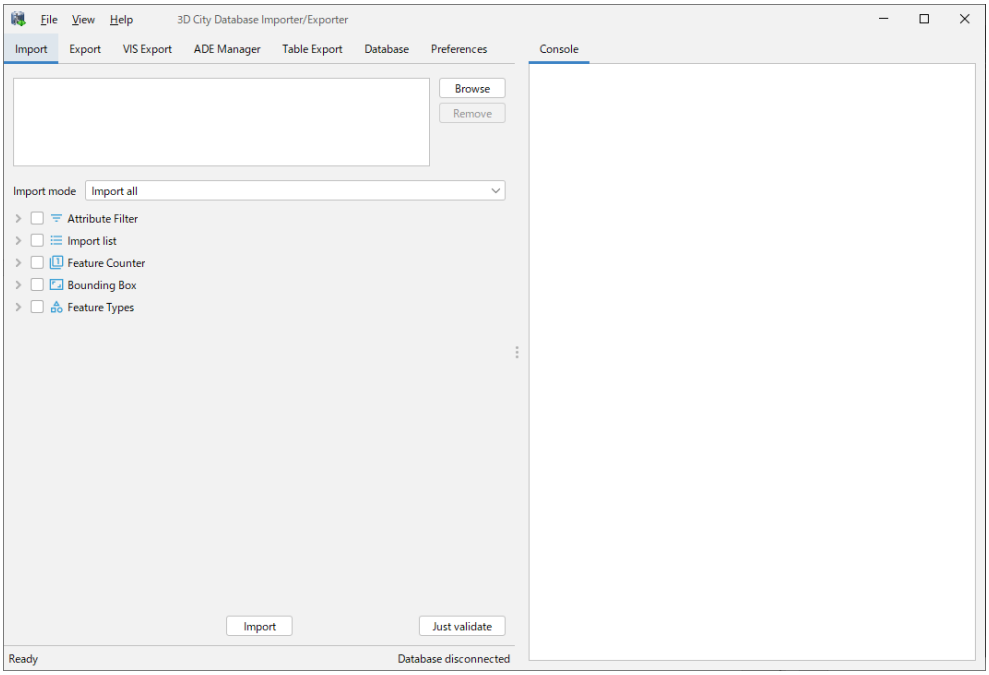


図 3-12 Importer/Exporter ツールの起動直後画面

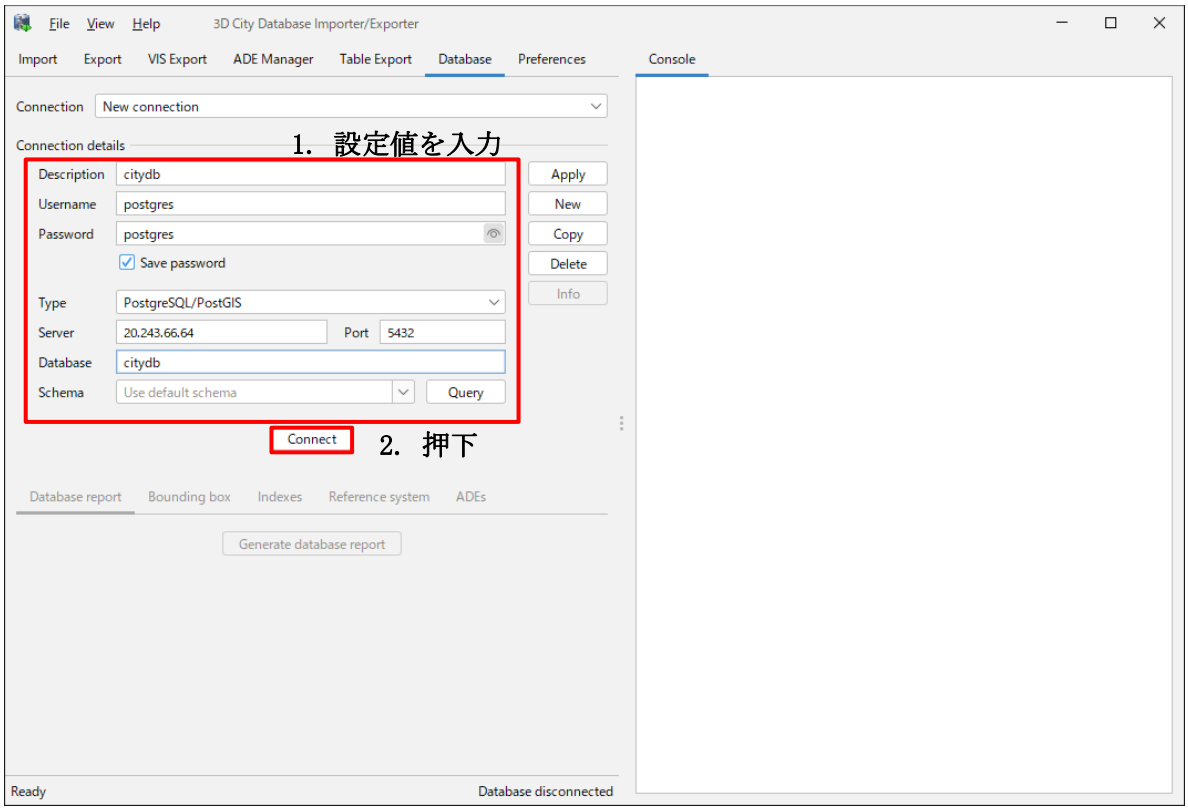
1. DB 設定

Database タブを選択し、DB 設定値を記入する。

表 3-2 Importer/Exporter ツールにおける DB 設定

項目	設定値	説明
Description	任意	接続の説明文。接続 DB が判別しやすい説明文が良い。
Username	postgres	PostgreSQL 接続用のユーザー名
Password	postgres	PostgreSQL 接続用のパスワード
Type	PostgreSQL/PostGIS	使用している DB の種別
Server	20.243.66.64	サーバーの IP アドレス
Port	5432	PostgreSQL が使用しているポート番号
Database	citydb	データベース名

接続前



接続後

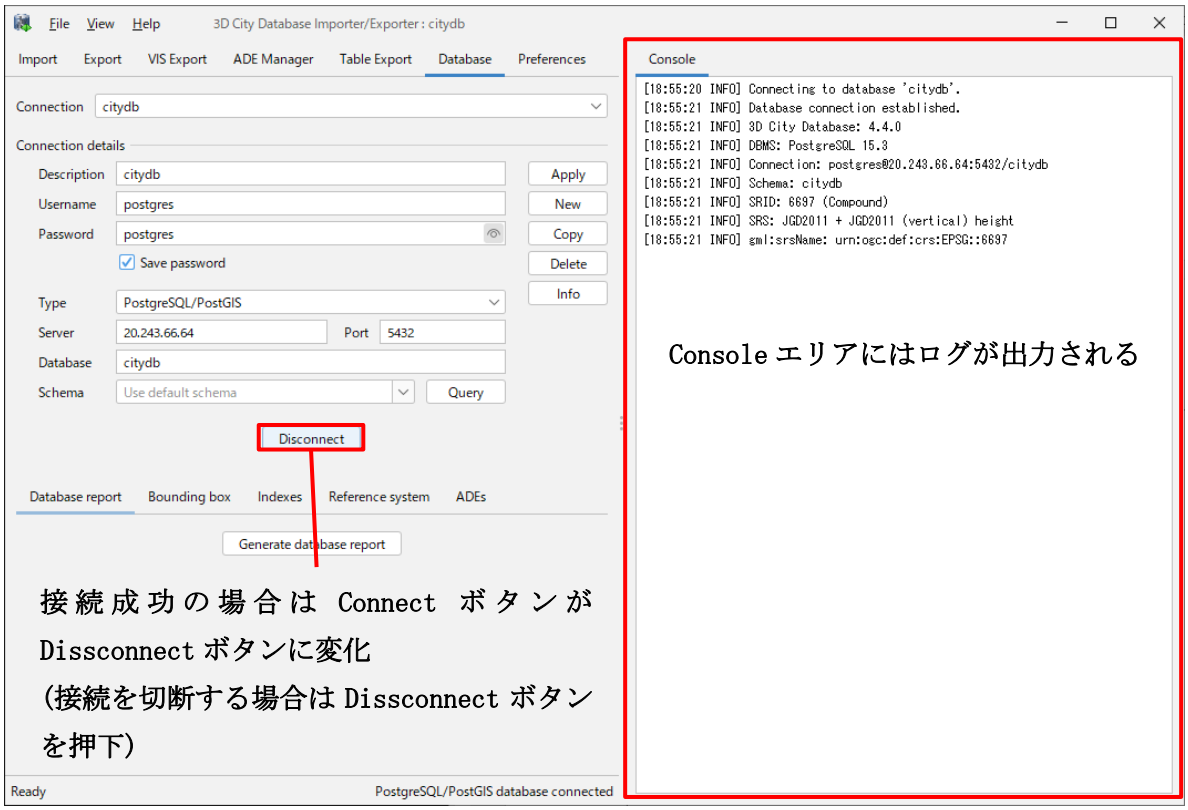


図 3-13 Importer/Exporter ツールでの DB 接続

3.3.2. A5:SQL Mk-2 での DB 接続

A5:SQL Mk-2 における DB 接続方法を以下に示す。

【A5:SQL Mk-2 における DB 接続方法】

1. A5:SQL Mk-2 の起動とワークスペース設定

解凍フォルダ内の A5M2.exe をダブルクリックする。

使用するワークスペースを選択して起動する。

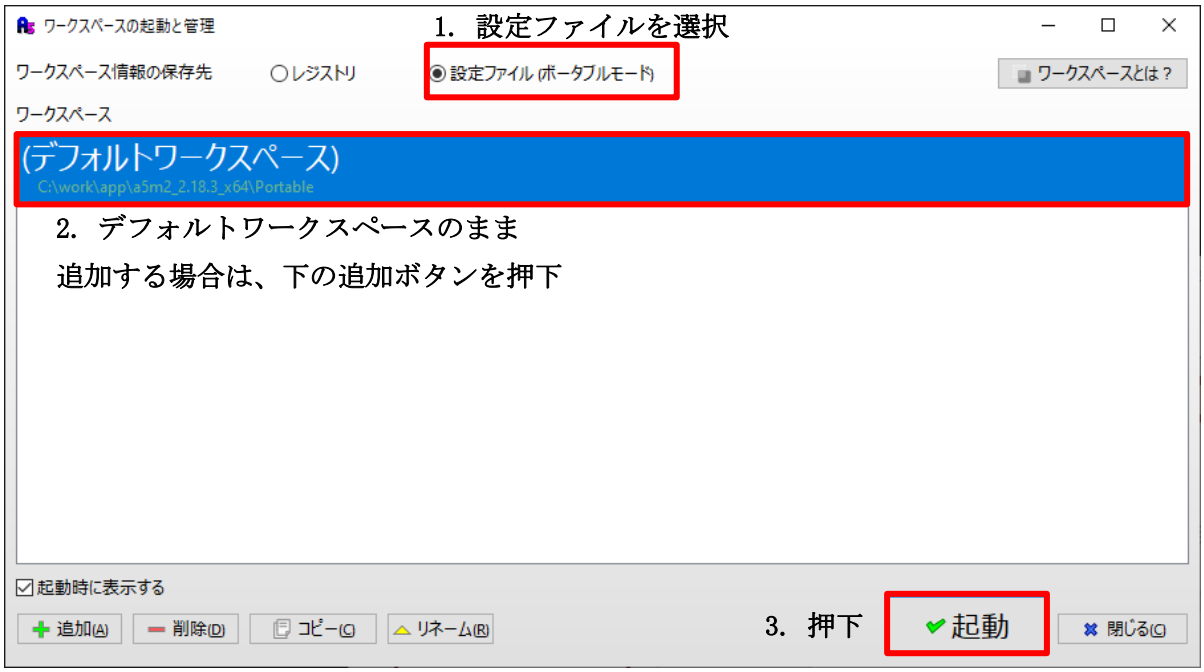


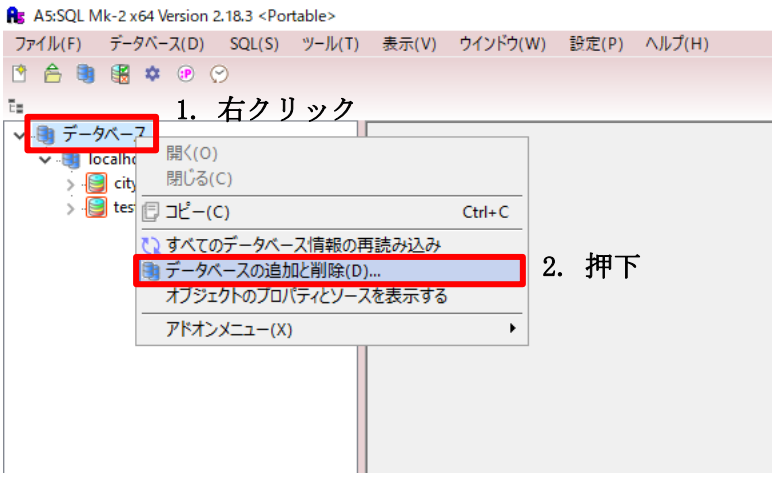
図 3-14 M5:SQL Mk-2 の起動

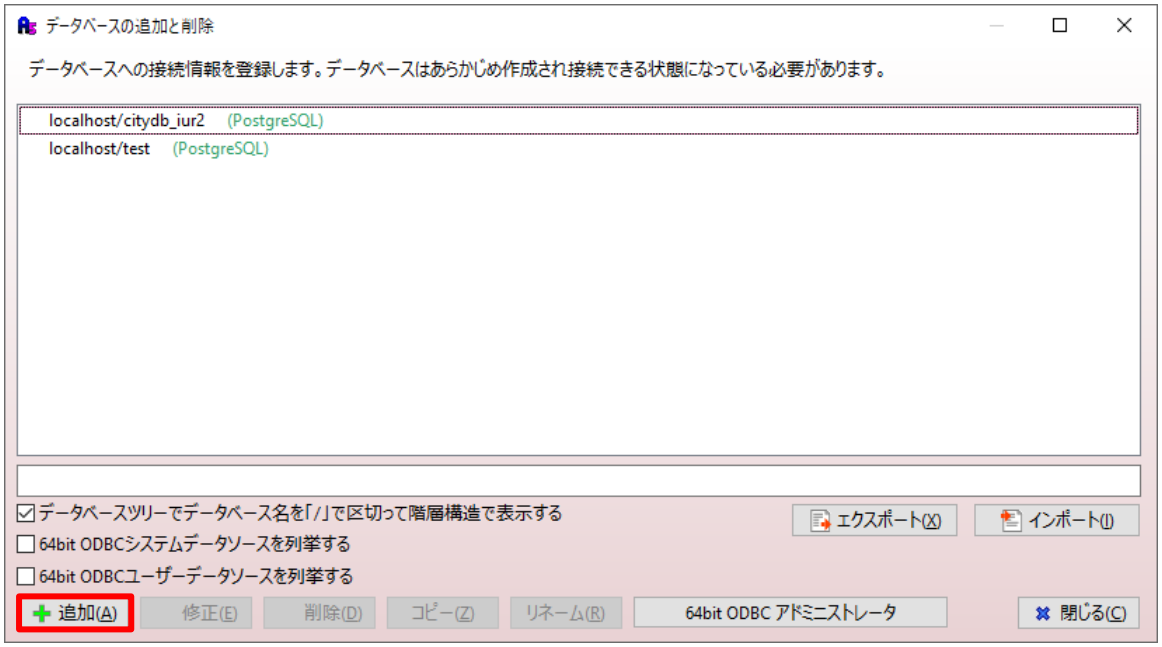
2. DB 登録

接続する DB を登録する。この作業は初回時のみの作業であり、既に DB を登録済みの場合は省略可能。DB 設定値は以下のとおり。

表 3-3 Importer/Exporter ツールにおける DB 設定

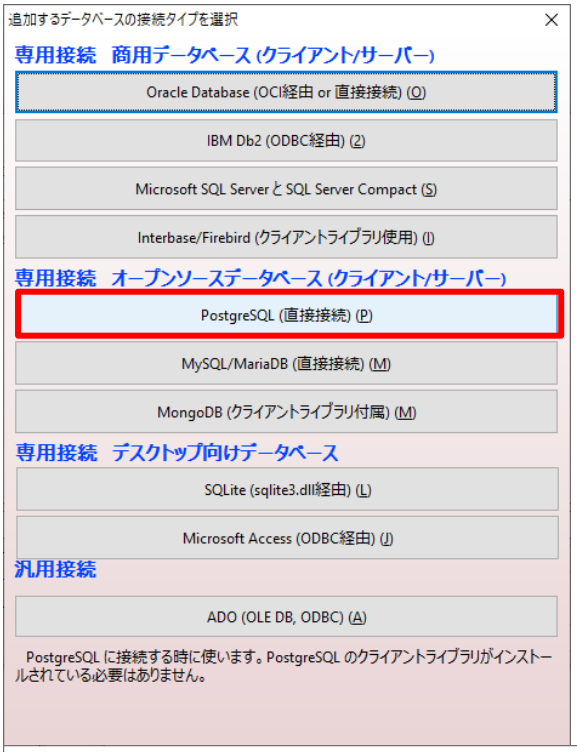
項目	設定値	説明
DB 種類	PostgreSQL	使用している DB の種別
サーバー名	20.243.66.64	サーバーの IP アドレス
ポート番号	5432	PostgreSQL が使用しているポート番号
データベース名	citydb	データベース名
ユーザーID	postgres	PostgreSQL 接続用のユーザー名
パスワード	postgres	PostgreSQL 接続用のパスワード
プロトコルバージョン	3.0 (PostgreSQL 7.4 ～)	PostgreSQL のプロトコルバージョン



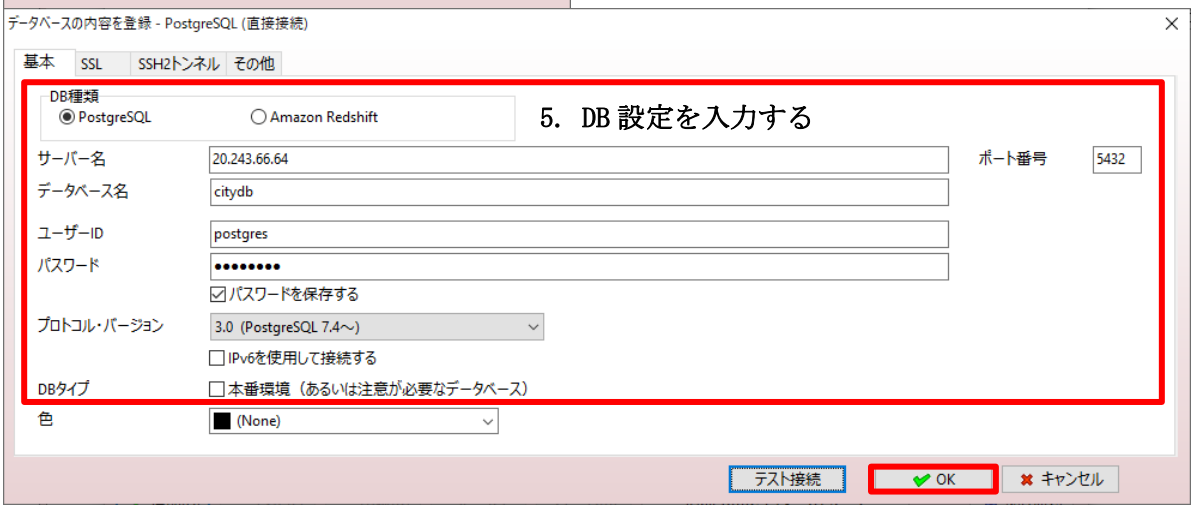


3. 押下

図 3-15 M5:SQL Mk-2 の DB 登録方法 (1/3)



4. PostgreSQL（直接接続）を選択



6. OK ボタンを押下(テスト接続してから OK ボタン押下でもよい)

7. 登録名を入力して、OK ボタンを押下
（デフォルト値でよければ、何もせずに OK ボタンを押下）

図 3-16 M5:SQL Mk-2 の DB 登録方法 (2/3)

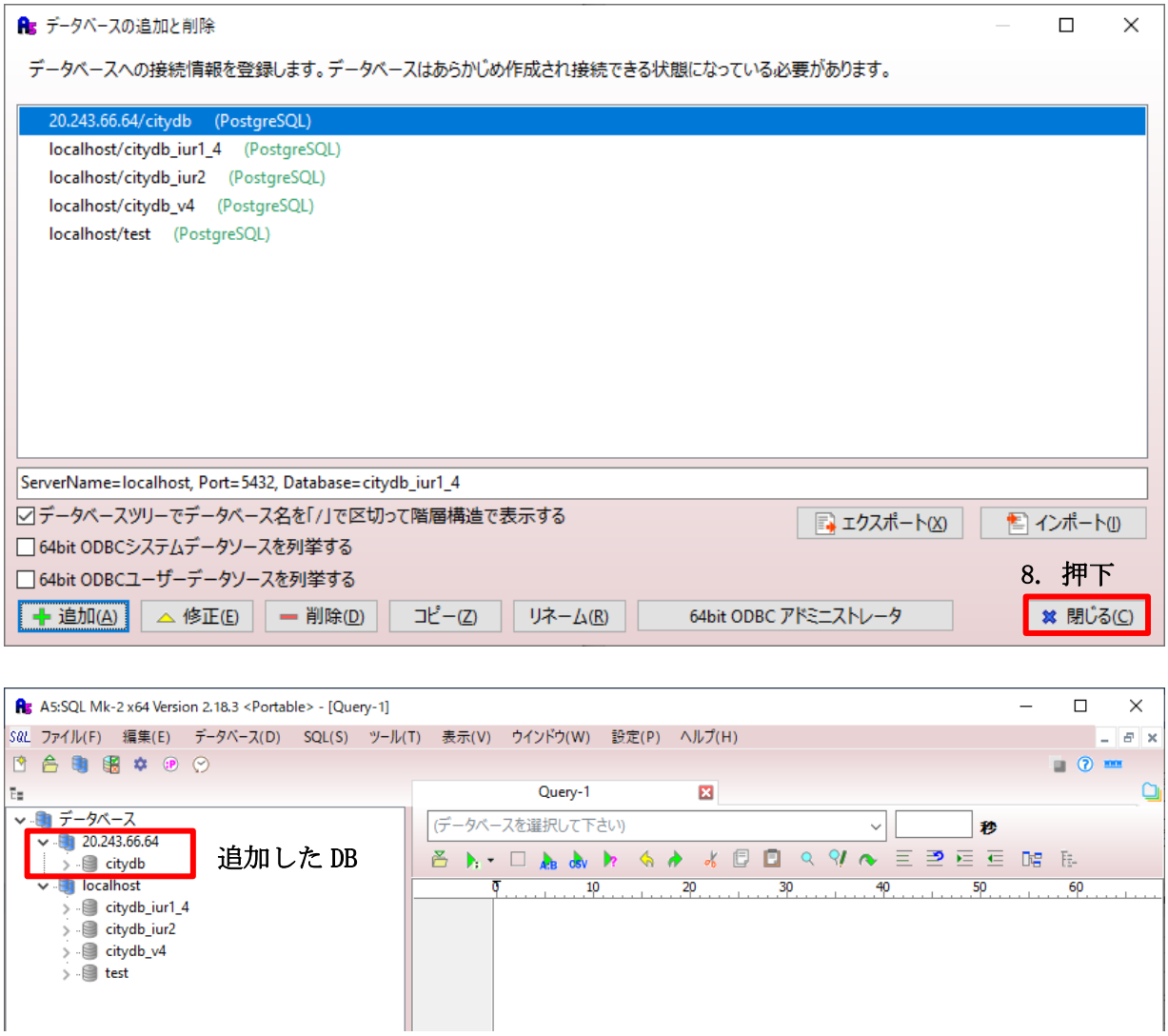


図 3-17 M5:SQL Mk-2 の DB 登録方法 (3/3)

3. DB 接続

DB 接続方法を下図に示す。

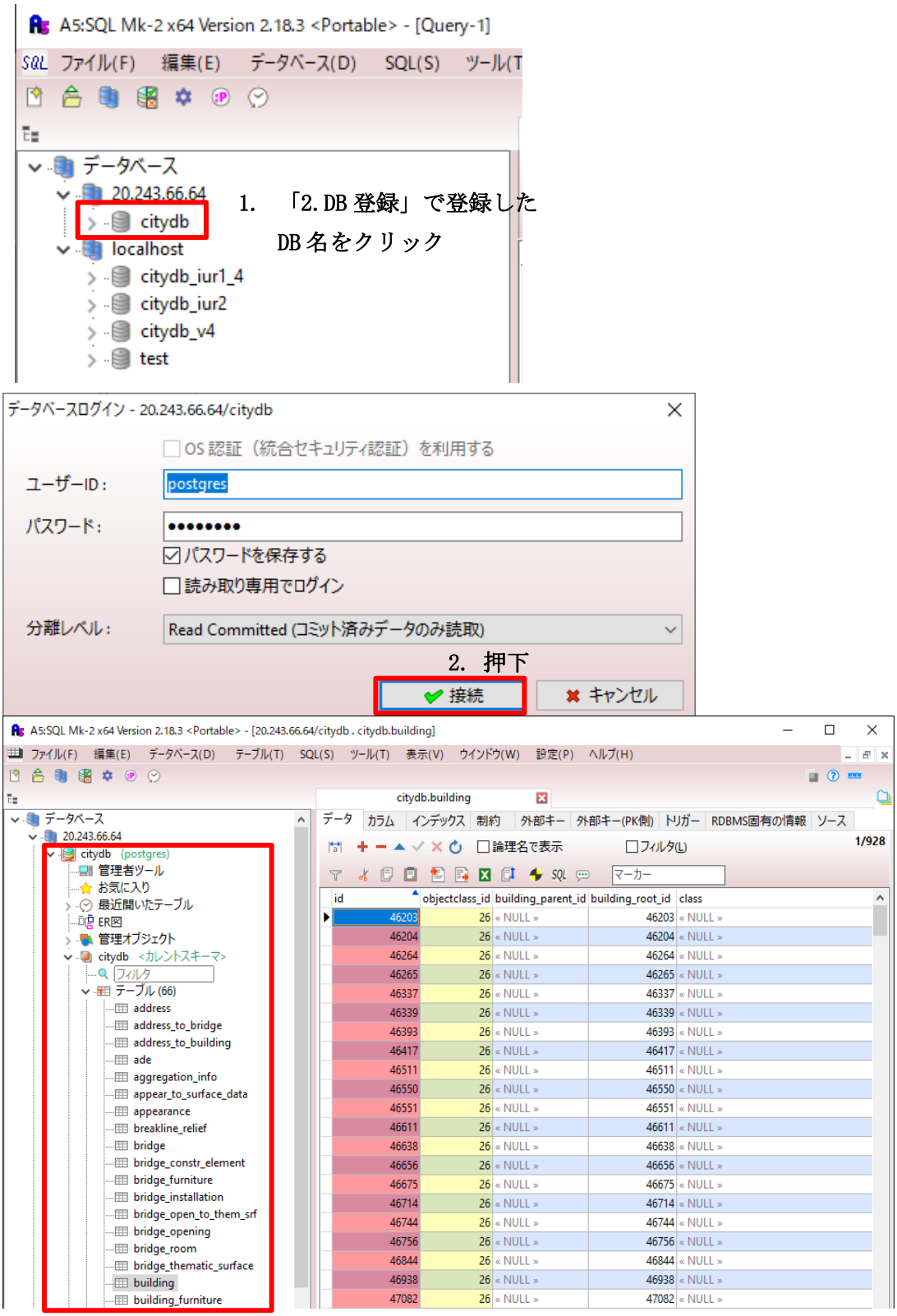


図 3-18 M5:SQL Mk-2 の DB 接続方法

3.4. CityGML データのインポート・エクスポート方法

3.4.1. CityGML データのインポート方法

インポート対象の CityGML データの座標は、（経度、緯度、高さ）の順に並んでいるものとする。座標が（緯度、経度、高さ）の順に並んでいても、DB へデータをインポートすることは可能であるが、3DCityDB-Web-Map-Client においてモデルデータを表示した際に、表示位置がずれる問題がある。

CityGML データのインポート方法を下図に示す。なお、データをインポートする DB への接続は確立しているものとする。

インポート可能なファイルは以下のとおり。

表 3-4 Importer/Exporter ツールにおける DB 設定

フォーマット	サポートバージョン	備考
CityGML（*.gml、*.xml）	ver.2.0、ver. 1.0、ver.0.4	
CityJSON（*.JSON、*.cityJSON）	ver.1.0.x	
gzip（*.gz、*.gzip）	-	CityGML 又は CityJSON ファイルの圧縮ファイル
ZIP（*.zip）	-	CityGML 又は CityJSON ファイルの圧縮ファイル

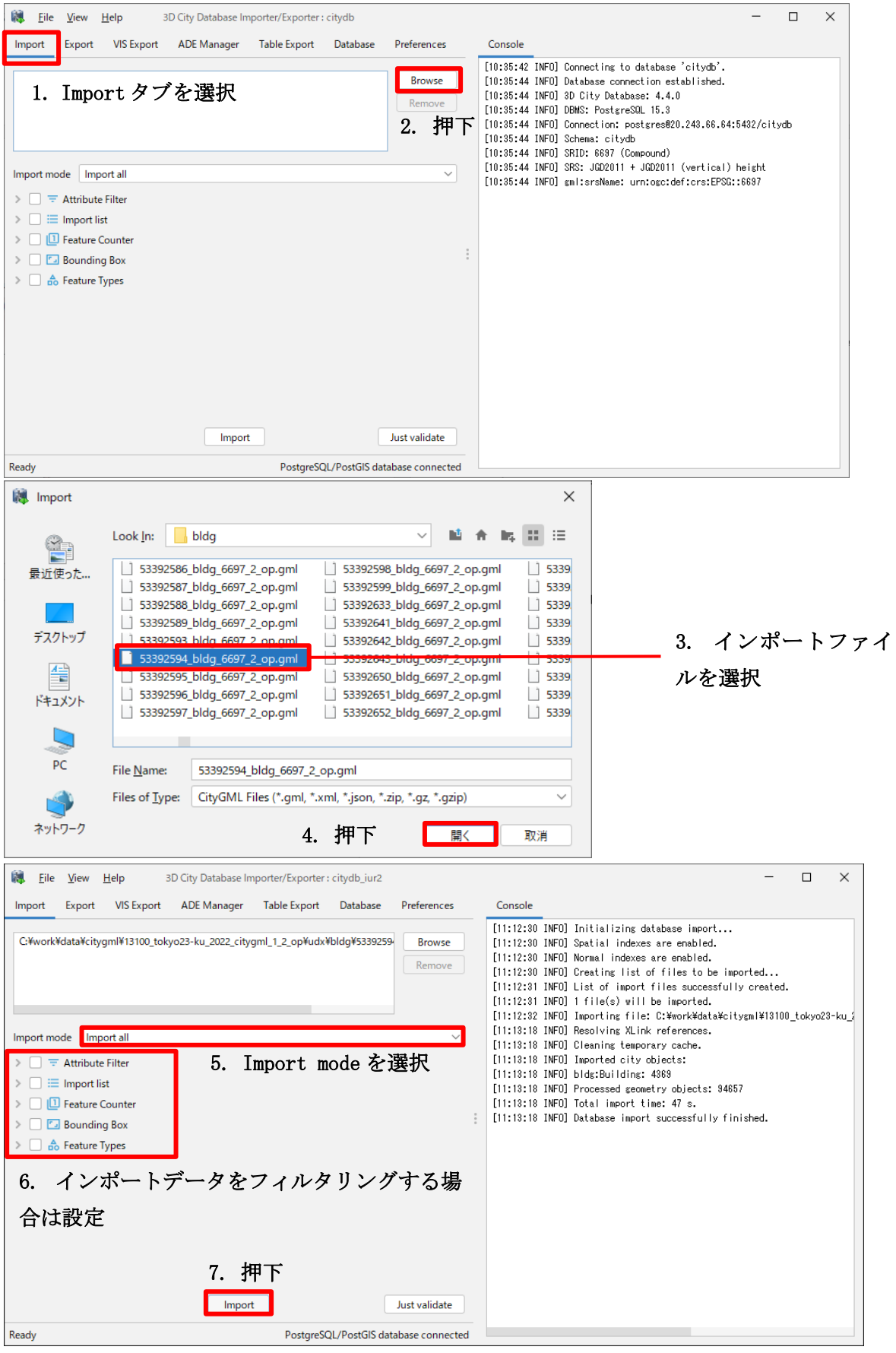


図 3-19 CityGML データのインポート方法

CityGML データのインポートモードに関しては下表のとおり。

表 3-5 Import mode

モード	説明
Import all	重複データの有無に関わらず、全てのデータをインポートする。
Skip existing	重複データをインポートする場合はスキップし、DB 内のデータを優先する。
Delete existing	重複データをインポートする場合は、DB 内の既存データを削除してからデータをインポートする。
Terminate existing	重複データをインポートする場合は、DB 内の既存データを終了扱いにしてからデータをインポートする。（既存データは DB に残る。）

インポート機能では、インポートする CityGML データに対してフィルタリングが可能である。各フィルタリング機能について以下に記載する。

【フィルタリング機能】

1. Attribute Filter

Attribute Filter は、オブジェクト識別子（Identifier）及びおよび gml:name をパラメータとして受け取り、それぞれの属性に一致する値を持つデータのみをインポートする。

Identifier は、複数の識別子をカンマ区切りで指定可能である。gml:name は、複数値の指定は対応していない。

図 3-20 Attribute Filter

gml:name で指定する検索文字列は以下のワイルドカード文字とエスケープ文字をサポートしている。

表 3-6 ワイルドカード文字とエスケープ文字

文字	種別	説明
*（アスタリスク）	ワイルドカード	0 文字以上を表す。
.（ドット）	ワイルドカード	1 文字を表す。
“”（セミコロン）	エスケープ	ワイルドカードをエスケープする場合は、セミコロンで囲む。 “*abc”とすると、gml:name が*abc と完全に一致しなければインポートしない。

2. Import list

Import list では、インポート時にインポート又はスキップする都市オブジェクトのリストを指定可能である。

インポートリストは、インポート又はスキップする都市オブジェクトの識別子を CSV ファイル形式で記載したデータである。

【インポートリスト　サンプル】

```
FEATURE_TYPE,GMLID
Building,bldg_e331c548-5531-47f9-8532-1c9f87abfaad
Building,bldg_e3c4f77a-d31a-42c2-a947-2c2c04395891
Building,bldg_24854724-efa3-4f0f-afb4-674f3145d897
```

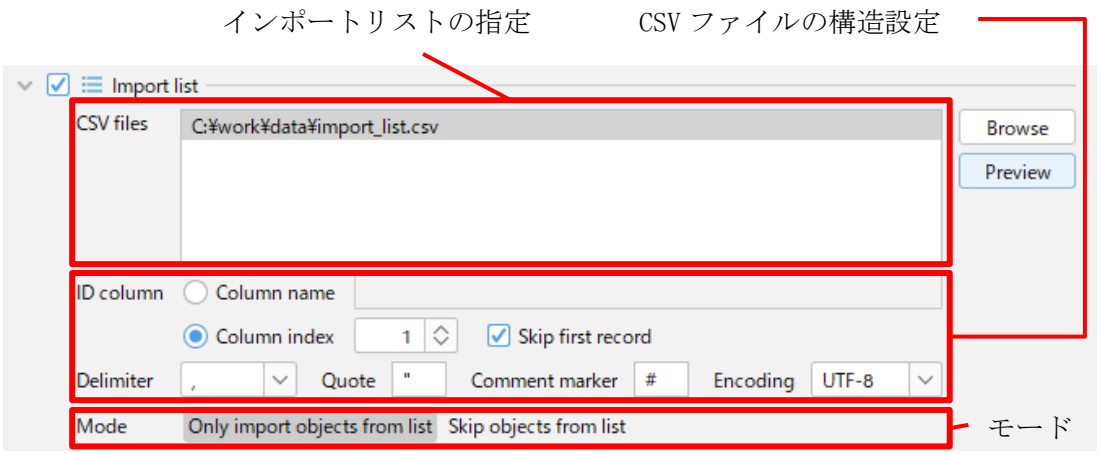


図 3-21 Import list

【インポートリストの指定】

Browse ボタン押下後に表示されるファイル選択ダイアログ、又はリストボックスエリアに CSV ファイルをドラッグ&ドロップすることで、インポートリストを指定することができる。なお、インポートリストは複数ファイルの指定が可能である。

Preview ボタンを押下すると、指定したインポートリストの最初の数行をプレビュー表示することが可能である。特定のファイル又はファイル群のプレビューを生成する場合は、表示したい CSV ファイルを選択してから、Preview ボタンを押下する。

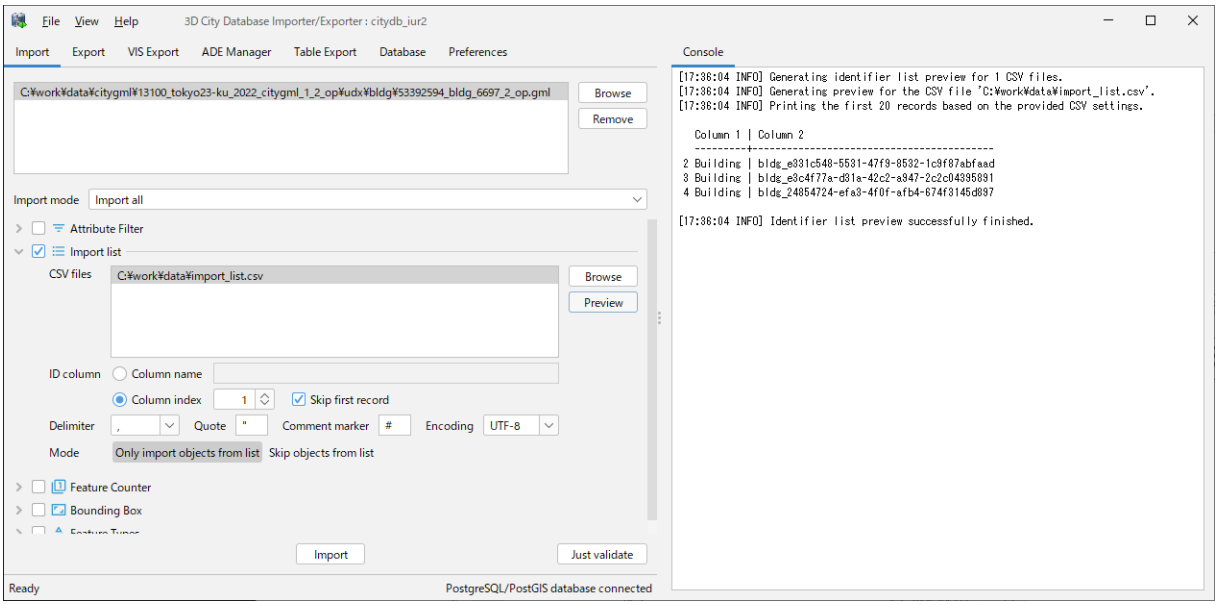


図 3-22 インポートリストのプレビュー結果

【CSV ファイル構造設定】

インポートリストの構造について設定するエリアである。複数のインポートリストを指定した場合、全てのインポートリストに対して入力した構造設定値が適用されるため、同一フォーマットのインポートリストを指定すること。

設定値について以下に示す。

表 3-7 設定値

設定	説明
ID column	ファイル内の識別子を保持する列を特定するに当たり、インポートリストがヘッダーを保持する場合は「Column name」において列名を指定する方法が使用可能である。その他、「Column index」において列のインデックス（1 始まり）を指定する方法も使用可能。「Column index」の場合、「Skip first record」の設定によって先頭行がヘッダーの場合は、先頭行を無視することができる。
Delimiter	CSV ファイルの区切り文字の指定。 デフォルトはカンマ区切りを想定している。
Quote	インポートリストの値が引用符で囲まれている場合、引用符として使用されている文字を指定する。 通常は、”（二重引用符）が使用される。
Comment marker	コメントとみなす文字を指定。
Encoding	ファイルの文字コードを指定。

【モード】

インポートリストの使用方法を設定する。

インポートリストで指定したデータのみをインストールする場合は「Only import objects from list」を、インポートリストで指定したデータを除外する場合は「Skip object from list」を選択する。

3. Feature Counter

Feature Counter は、インポートする最上位地物（建物、交通、土地利用等）の数を制限する。「Count」にはインポートする最上位地物数を、「Start index」には全最上位地物集合における、インポートを開始するインデックス番号（0 開始）を指定する。

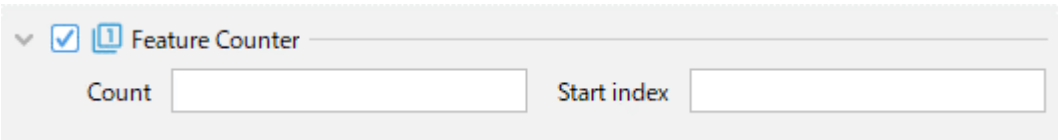


図 3-23 Feature Counter

4. Bounding Box

Bounding Box では、左下隅（x_{min}, y_{min}）右上隅（x_{max}, y_{max}）の座標値で与えられる 2D バウンディングボックスを入力とし、指定されたバウンディングボックスと重畳、又はバウンディングボックスの内側にある地物をインポートする。

「All overlapping features」を選択した場合はバウンディングボックスと重畳、「Only features inside」を選択した場合はバウンディングボックスの内側にある地物をインポートする。

「Reference system」は座標系を選択する部分であり、「Same as in database」を選択した場合は EPSG:6697（JGD2011 + JGD2011 (vertical) height）となる。

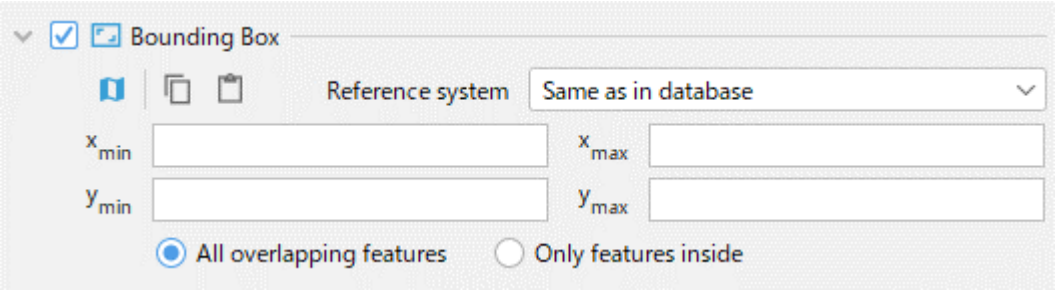


図 3-24 Bounding Box

バウンディングボックスの座標の指定は、マップウィンドウから設定することも可能である。

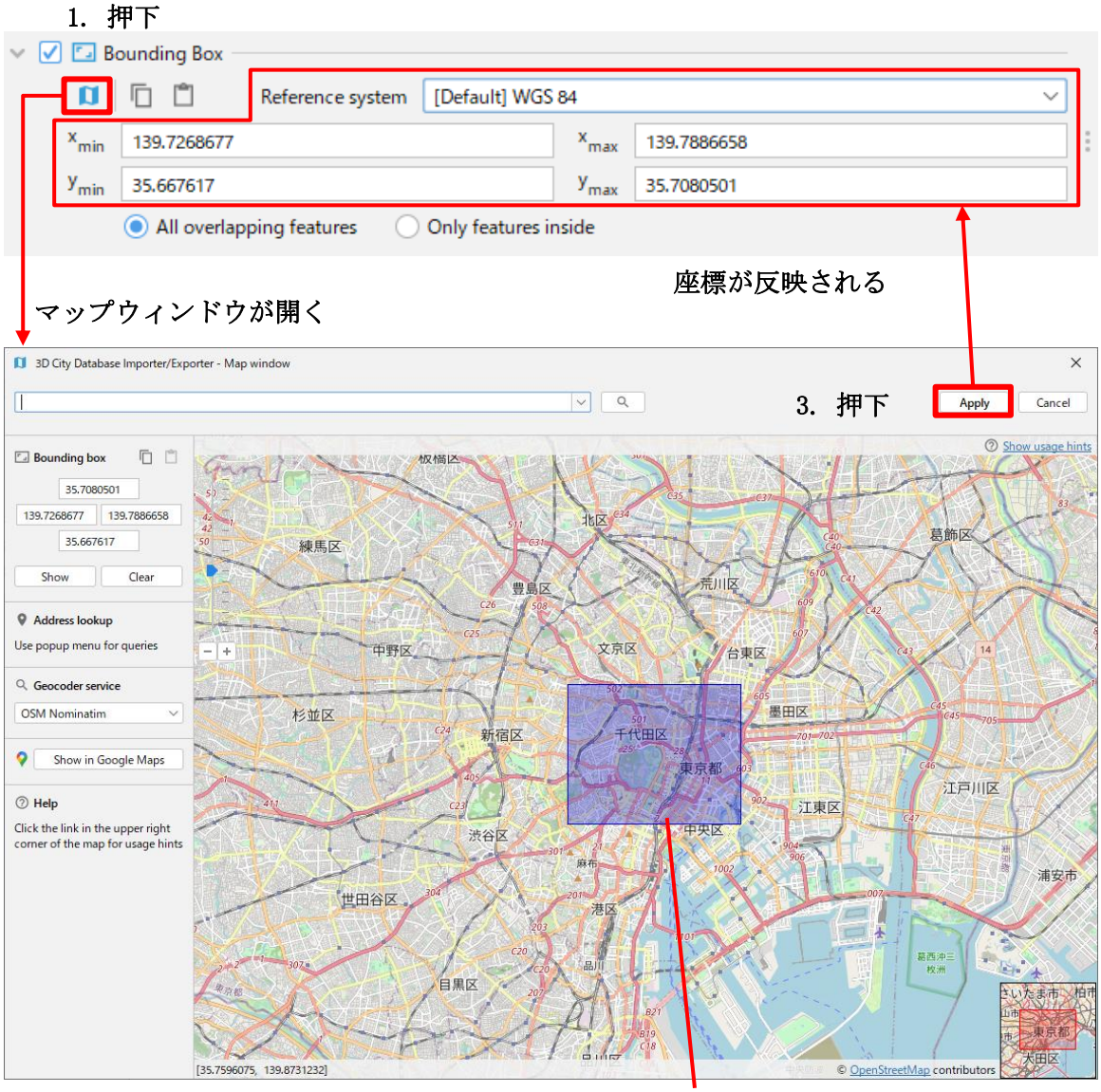


図 3-25 マップウィンドウによるバウンディングボックスの指定

5. Feature Types

Feature Types は、チェックボックスを有効にした地物のみをインポートする。

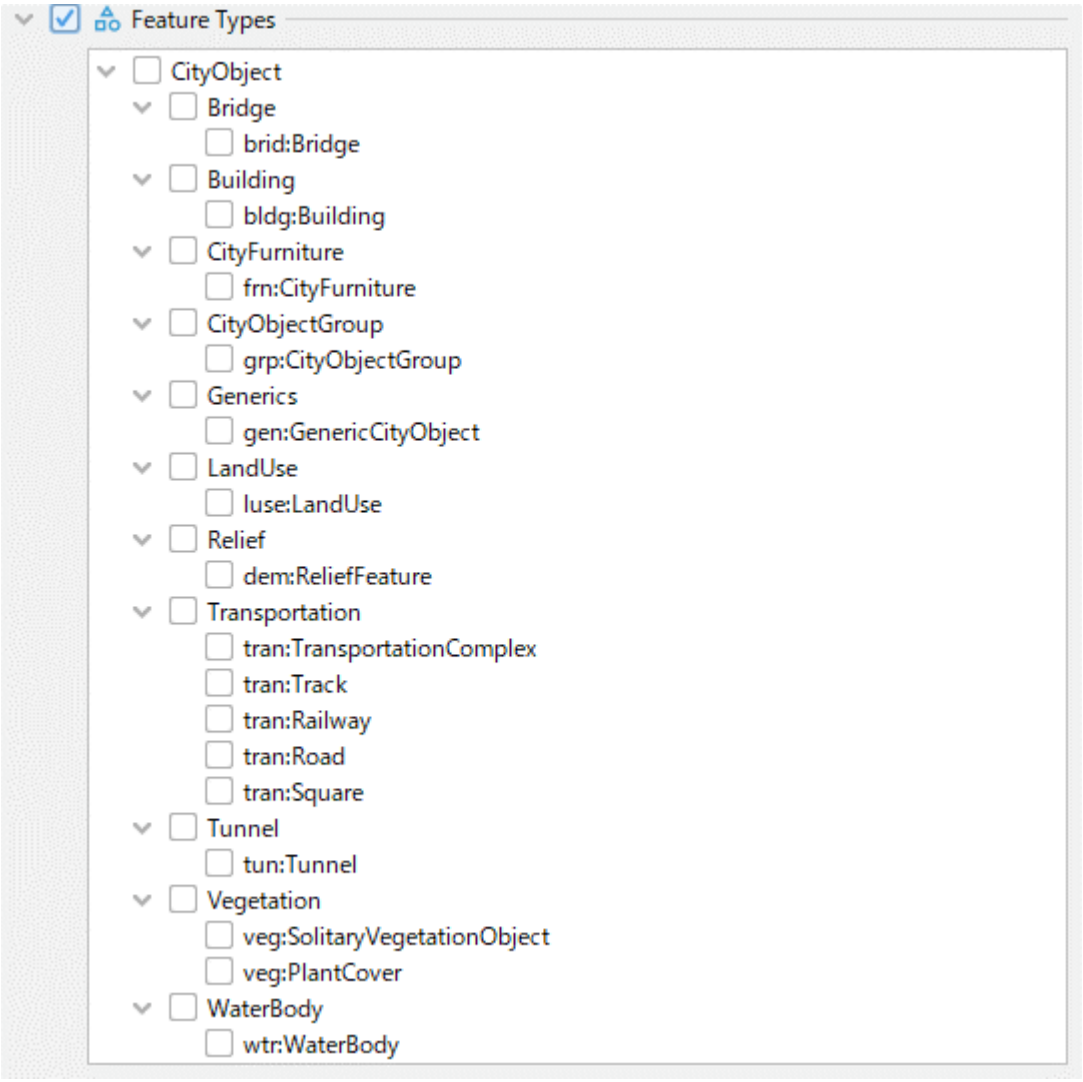


図 3-26 Feature Types

3.4.2. CityGML データのエクスポート方法

CityGML データのエクスポート方法を下図に示す。なお、エクスポート対象のデータを保持する DB への接続は確立しているものとする。

エクスポート可能なファイルは以下のとおり。

表 3-8 エクスポート対象フォーマット

フォーマット	サポートバージョン	備考
CityGML (*.gml、*.xml)	ver.2.0、ver.1.0	
CityJSON (*.JSON、*.cityJSON)	ver.1.0.x	
gzip (*.gz、*.gzip)	-	CityGML ファイルの圧縮ファイルを出力する。
ZIP (*.zip)	-	CityGML ファイルの圧縮ファイルを出力する。

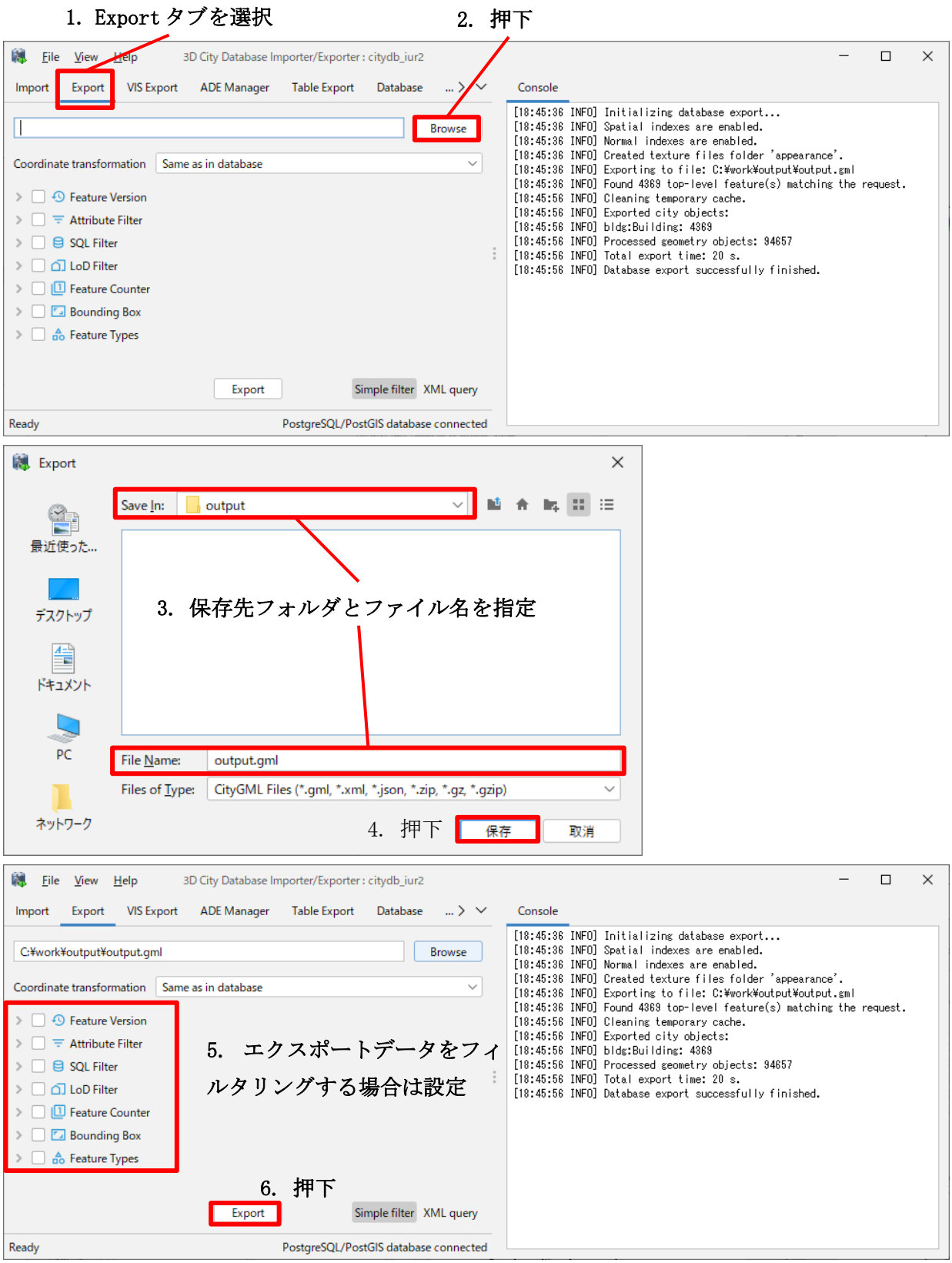


図 3-27 CityGML データのエクスポート

エクスポート機能では、エクスポートする CityGML データに対してフィルタリング可能である。各フィルタリング機能については以下に記載する。

【フィルタリング機能】

1. Feature Version

3DCityDB では、同一地物を複数バージョン（異なる時点）で管理することが可能である。したがって、データをエクスポートする際にどのバージョンの地物を出力するか選択することが可能である。

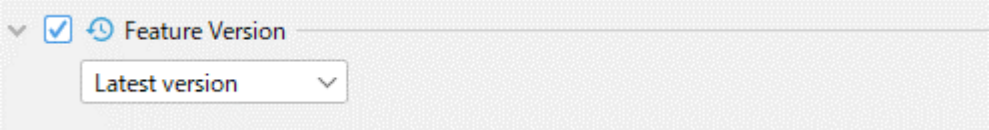


図 3-28 Feature Version

表 3-9 Feature Version

Feature Version	説明
Latest version	DB 上で終了フラグがついていない最新バージョンの地物をエクスポートする。
Valid version	指定されたタイムスタンプ又は時間範囲で有効であった地物をエクスポートする。
Terminated version	DB 上で終了フラグがついている地物をエクスポートする。 全ての終了地物をエクスポートするか、指定されたタイムスタンプで終了した地物のみをエクスポートするかを選択可能。

2. Attribute Filter

Attribute Filter では、オブジェクト識別子（Identifier）、gml:name、citydb:lineage の値を設定可能である。

Identifier は、複数の識別子をカンマ区切りで指定可能である。gml:name と citydb:lineage は、複数値の指定は対応していない。

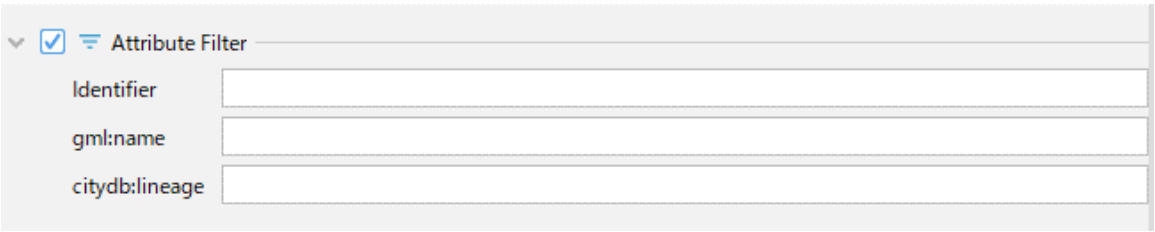


図 3-29 Attribute Filter

gml:name と citydb:lineage で指定する検索文字列はワイルドカード文字とエスケープ文字をサポートしている。詳細については、インポート機能の Attribute Filter と同様であるため、「3.4.1CityGML データのインポート方法」の「Attribute Filter」を参照。

3. SQL Filter

SQL Filter では、ユーザー定義の SELECT 文を用いてエクスポートする地物を選択可能である。なお、SQL 文は、必ず cityobject テーブルの id データを結果として返す必要がある点に注意すること。

以下に SQL 文のサンプルを記載する。

【SQL 文サンプル】

地区計画属性が銀座地区の地物を取得する SQL 文

select cityobject_id from cityobject_genericattrib where attrname='地区計画' and strval='銀座地区'

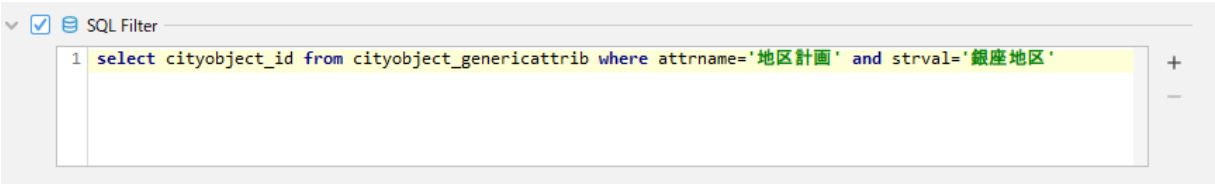


図 3-30 SQL Filter

4. LoD Filter

LoD Filter では、チェックボックスを有効にした詳細度の地物をエクスポートする。複数の詳細度を選択した場合は、Filter mode において複数条件の評価方法を指定する必要がある。

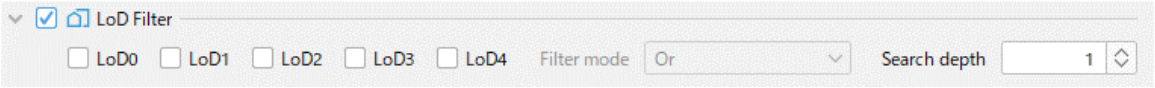


図 3-31 LoD Filter

表 3-10 Filter mode

Filter mode	説明
Or	選択された詳細度のうち、1 つ以上の選択された詳細度のモデルを保持する地物をエクスポートする。
And	選択された全ての詳細度のモデルを保持する地物をエクスポートする。
Minimum LoD	Or モードの特別バージョン。 一致する詳細度から最も低い詳細度のモデルのみをエクスポートする。
Maximum LoD	Or モードの特別バージョン。 一致する詳細度から最も高い詳細度のモデルのみをエクスポートする。

例えば、CityGML の bldg:Building は、自身の bldg:lod2Solid や gldg:lod2MultiSurface を通して LoD2 ジオメトリを提供する必要がなく、代わりに bldg:WallSurface や bldg:RoofSurface のように LoD2 表現を持つネストされた境界面のリストを保持することが可能である。このような場合に対応するために、Search depth においてネストした機能を何階層まで考慮するかを指定可能である。Search depth が 1 の場合は、最上位機能（自身）と 1 階層下の子機能に対して探索を行う。ワイルドカードの*（アスタリスク）を指定した場合は、深度に関係なく全てのオブジェクトに対して探索を行う。

5. Feature Counter

インポート機能の Feature Counter と同一仕様のため、「3.4.1CityGML データのインポート方法」の「Feature Counter」を参照。

6. Bounding Box

Bounding Box では、左下隅（x_{min}, y_{min}）右上隅（x_{max}, y_{max}）の座標値で与えられる 2D バウンディングボックスを入力とし、指定されたバウンディングボックスと地物の包絡線が重畳、又はバウンディングボックスの内側にある場合に地物をエクスポートする。

「All overlapping features」を選択した場合はバウンディングボックスと重畳、「Only features inside」を選択した場合はバウンディングボックスの内側にある地物をエクスポートする。

「Tile into rows」を選択した場合、バウンディングボックスを指定された行数と列数に分割したタイルごとのファイルに地物をエクスポートする。

「Reference system」は座標系を選択する部分であり、「Same as in database」を選択した場合は EPSG:6697（JGD2011 + JGD2011 (vertical) height）となる。

バウンディングボックスの指定方法は、インポート機能の Bounding Box と同一仕様のため、「3.4.1CityGML データのインポート方法」の「Bounding Box」を参照。

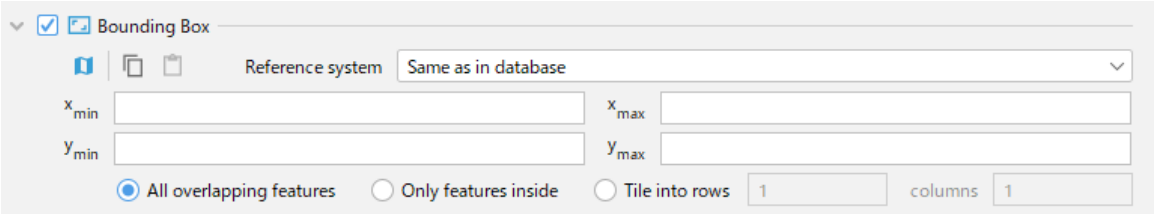


図 3-32 Bounding Box

7. Feature Types

インポート機能の Feature Types と同一仕様のため、「3.4.1CityGML データのインポート方法」の「Feature Types」を参照。

3.5. 3DCityDB-Web-Map-Client の利用方法

3DCityDB-Web-Map-Client は、3DCityDB に保存しているデータを閲覧するためのビューアである。

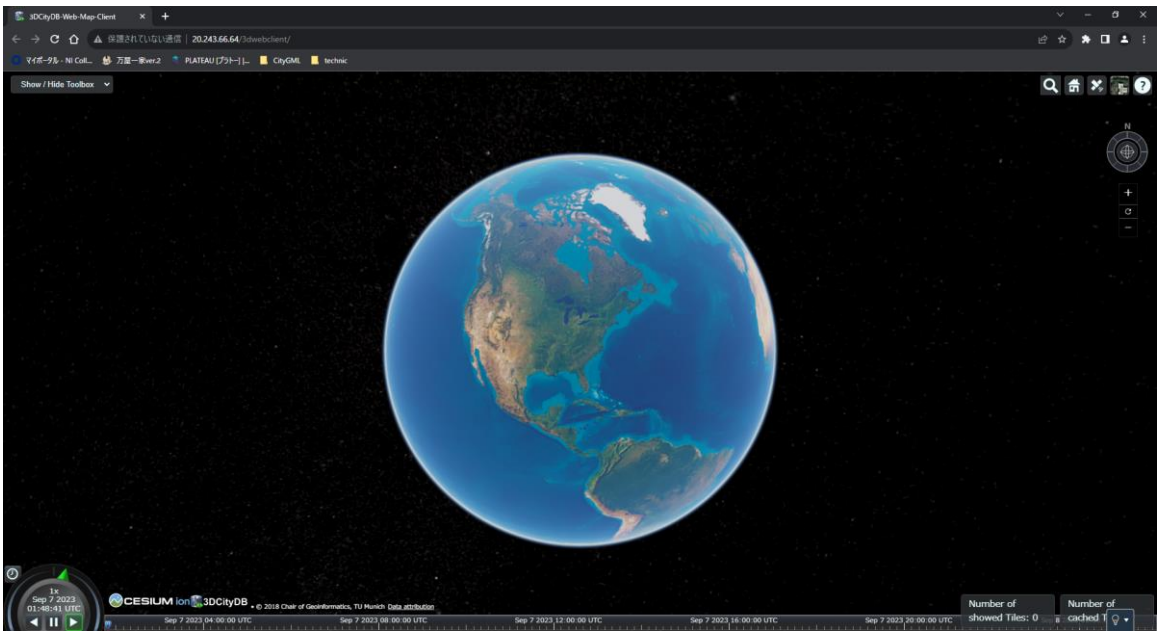


図 3-33 3DCityDB-Web-Map-Client

3DCityDB-Web-Map-Client の利用方法について以下に示す。

【3DCityDB-Web-Map-Client の利用方法】

1. 3DCityDB-Web-Map-Client の起動

Web ブラウザを起動し、アドレスバーにサーバー環境の URL を指定する。

2. データ設定

Add / Configure Layer に表示するデータの設定値を入力し、レイヤーを登録する。

設定値とレイヤー登録方法については以下を参照。

表 3-11 Add / Configure Layer の設定値

項目	設定値
URL	http://20.243.66.64/data/export_tokyo_bldg/output_extruded_MasterJSON.JSON
Name	tokyo（任意名）
Layer data type	COLLADA/KML/glTF（デフォルト設定）
Load via proxy	チェックなし（デフォルト設定）
KML clamp to ground	チェックあり（デフォルト設定）
glTF version	2.0（デフォルト設定）
thematicDataUrl	http://20.243.66.64:3000/building_attr_view
Thematic Data Source	PostgreSQL REST API
Table type	All object attributes in one row（デフォルト設定）
cityobjectsJsonUrl	空欄（デフォルト設定）
minLodPixels	空欄（デフォルト設定）
maxLodPixels	空欄（デフォルト設定）
MaxCountOfVisibleTiles	200（デフォルト設定）
MaxSizeOfCashedTiles	200（デフォルト設定）

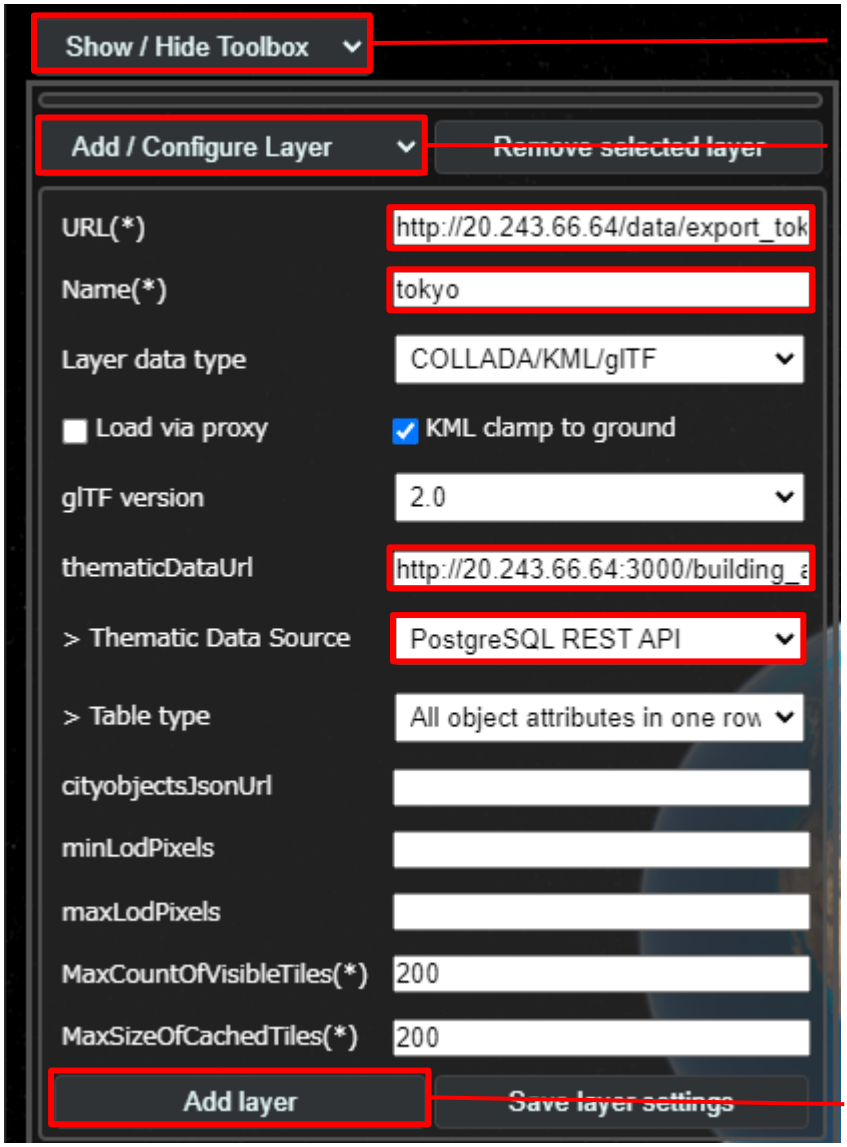
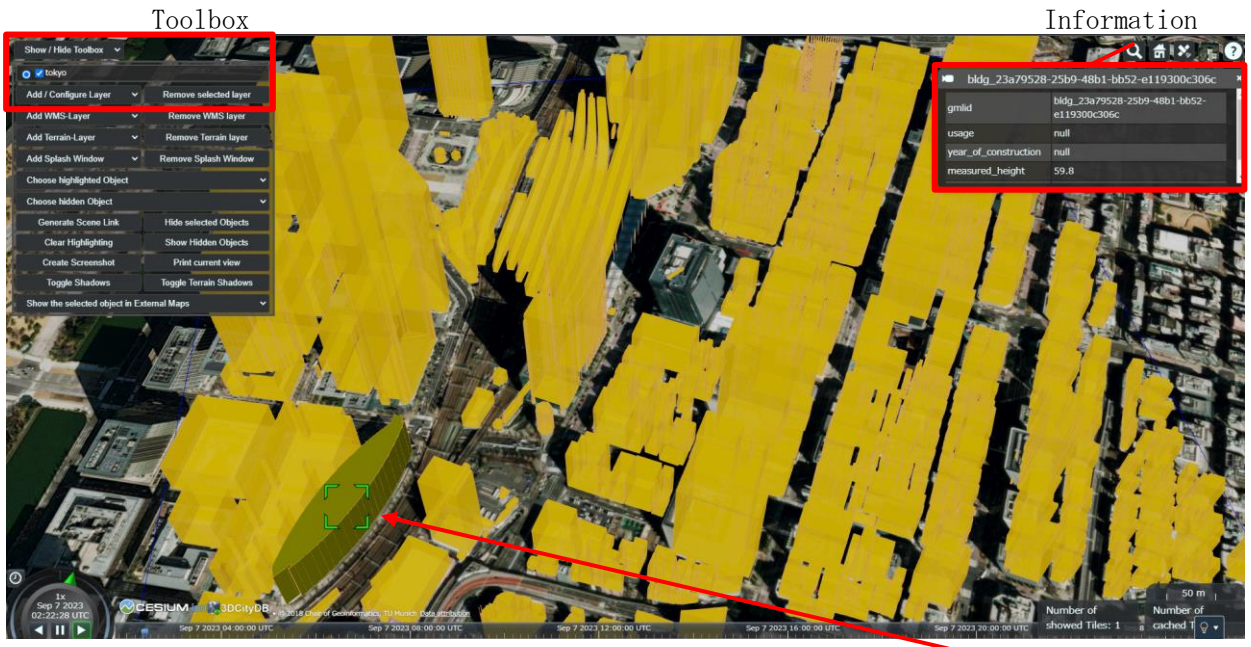


図 3-34 Add / Configure Layer の登録

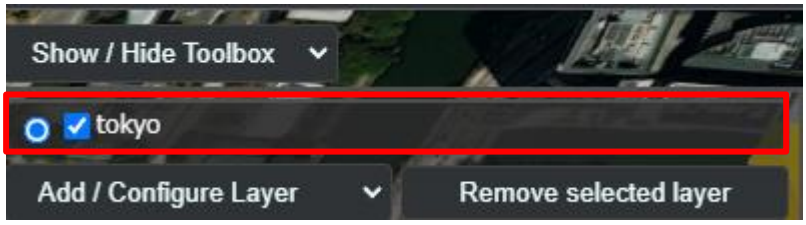
3. データの閲覧

Add Configure Layer の登録が終了すると、Toolbox 内に登録レイヤーが表示される。レイヤー名をダブルクリックするとモデルが存在するエリアに視点が移動する。

表示されているモデルを左クリックによって選択すると、右上にモデルの属性情報が表示される。



Toolbox 拡大



登録レイヤー

Information 拡大

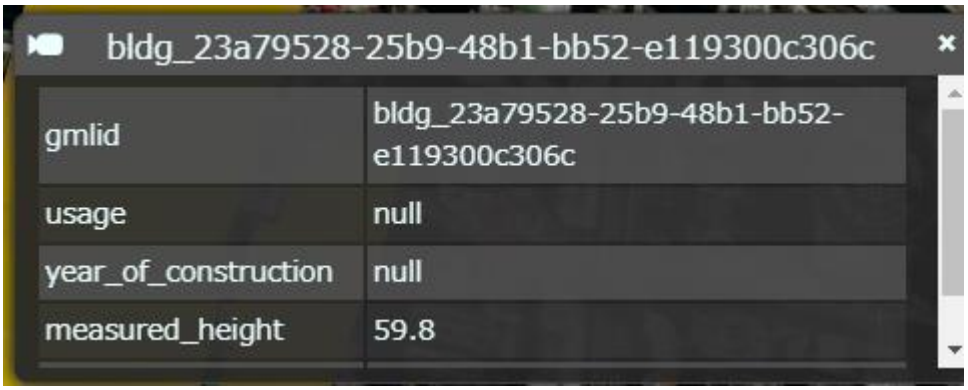


図 3-35 データ閲覧

3.6. A5:SQL Mk-2 によるデータ更新

A5:SQL Mk-2 を利用した DB のデータ更新方法を以下に示す。なお、A5:SQL Mk-2 は作業対象の DB への接続が確立しているものとする。

【A5:SQL Mk-2 によるデータ更新方法】

1. SQL 文の作成

更新対象のテーブル名を右クリックし、コンテキストメニューから「SQL の生成...」を選択する。表示されるダイアログを用いて UPDATE 文のフォーマットを作成する。作成した UPDATE 文において、接頭字が:(コロン)のパラメータを変更後の値や条件値などに書き換える。本資料では、building テーブルの id=46204 のレコードに対して、measured_height を 10.0 に更新する。

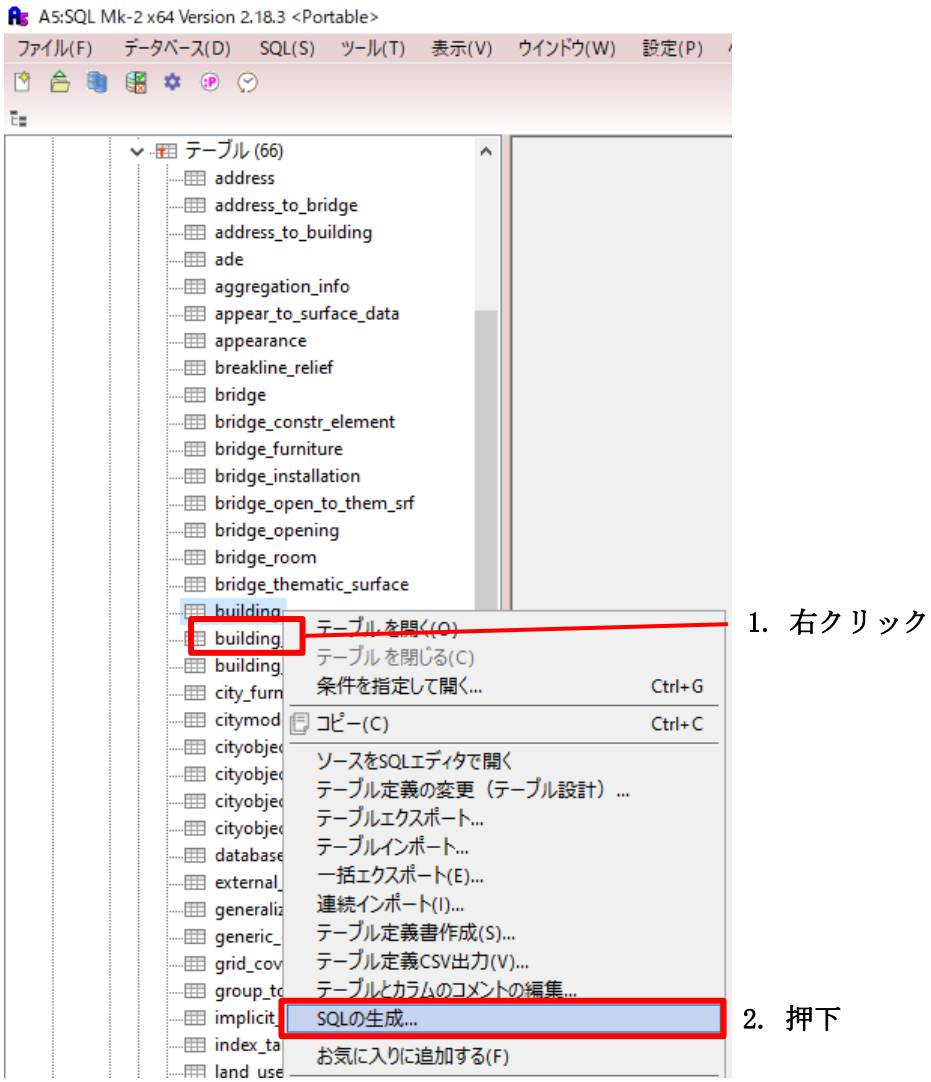


図 3-36 SQL（UPDATE）文の作成（1/2）

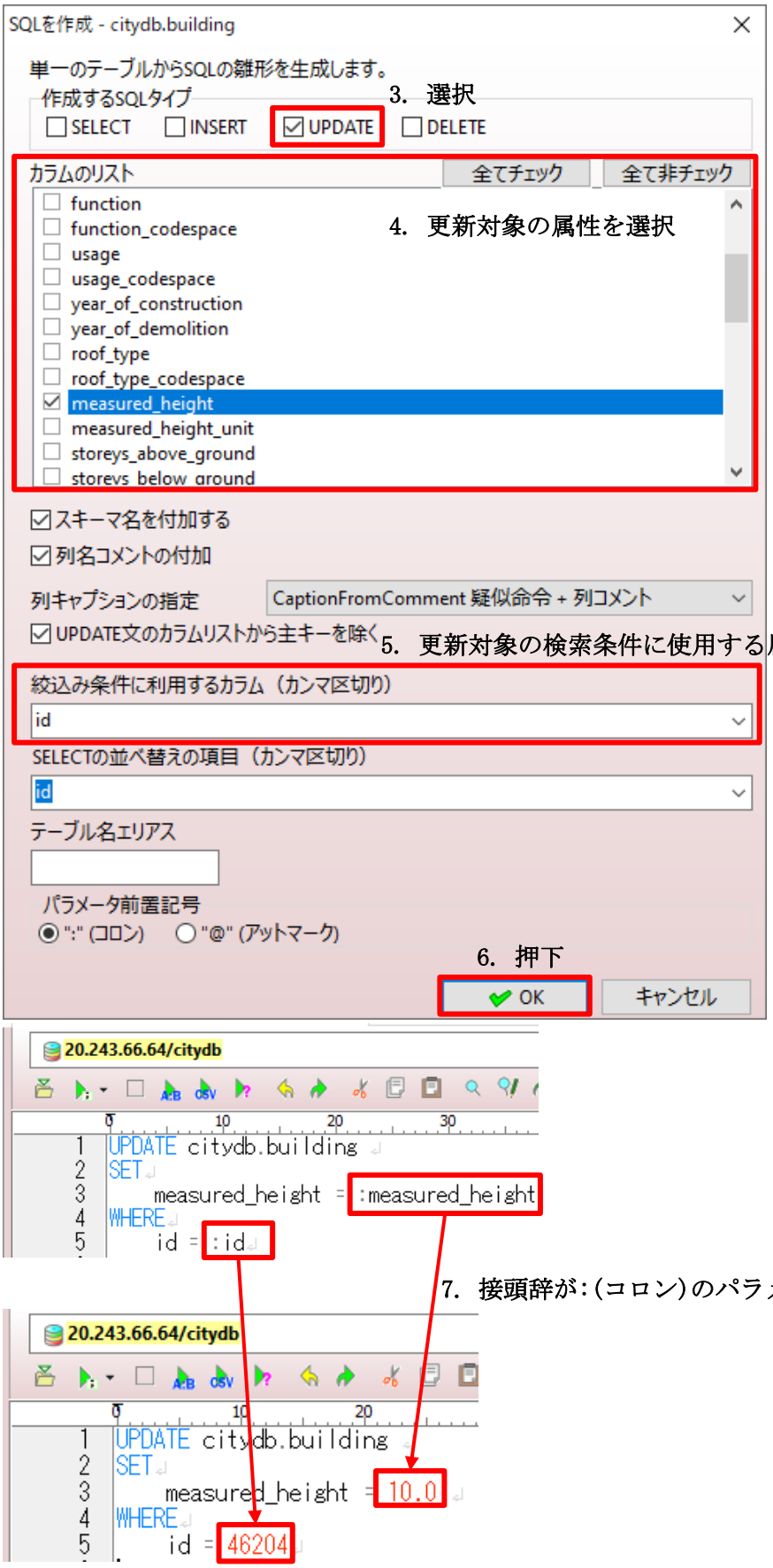


図 3-37 SQL（UPDATE）文の作成（2/2）

2. SQL の実行

作成した UPDATE 文を実行し、DB を更新する。なお、変更前後のデータの確認方法、及び SQL の実行方法を以下に示す。

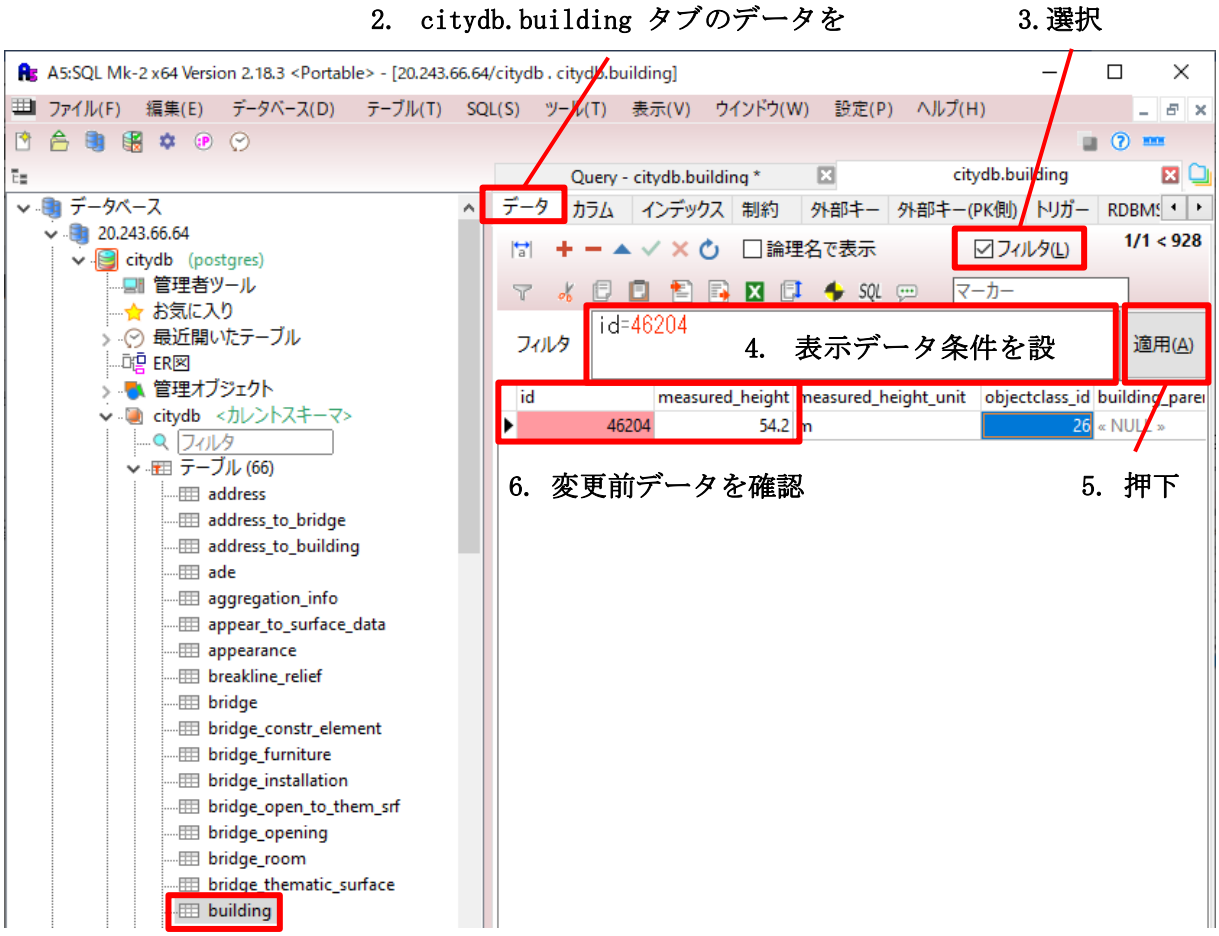
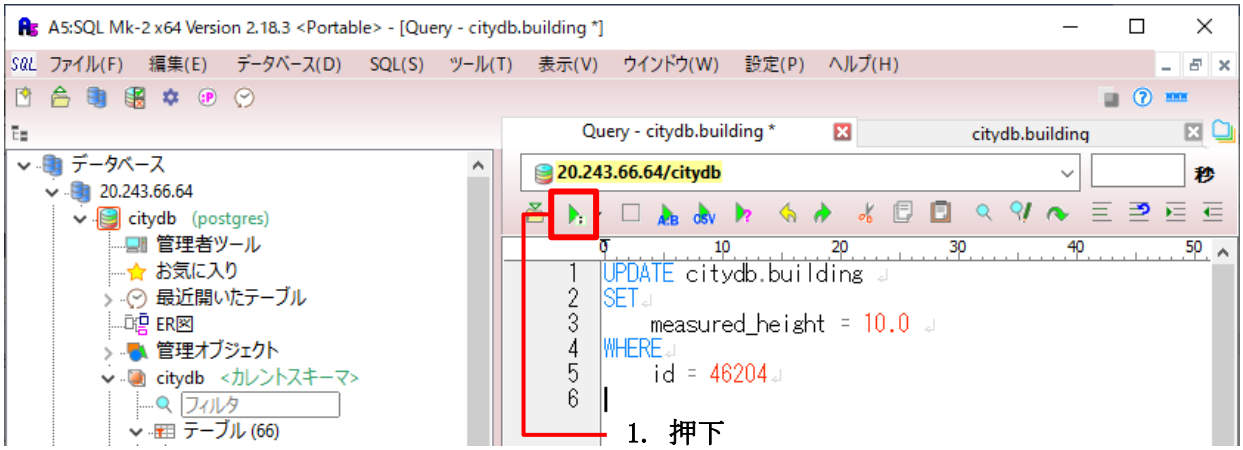


図 3-38 変更前データの確認方法



2. 更新メッセージが表示される

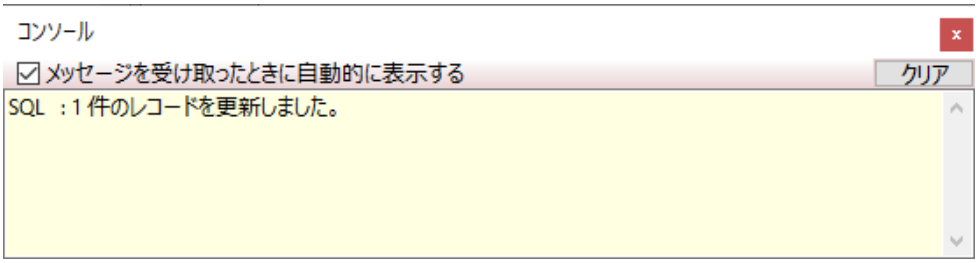


図 3-39 SQL の実行方法

なお、変更前データの確認で使用したタブが残っている場合は、下図のようにデータ更新ボタンを押下すると最新のデータが表示される。

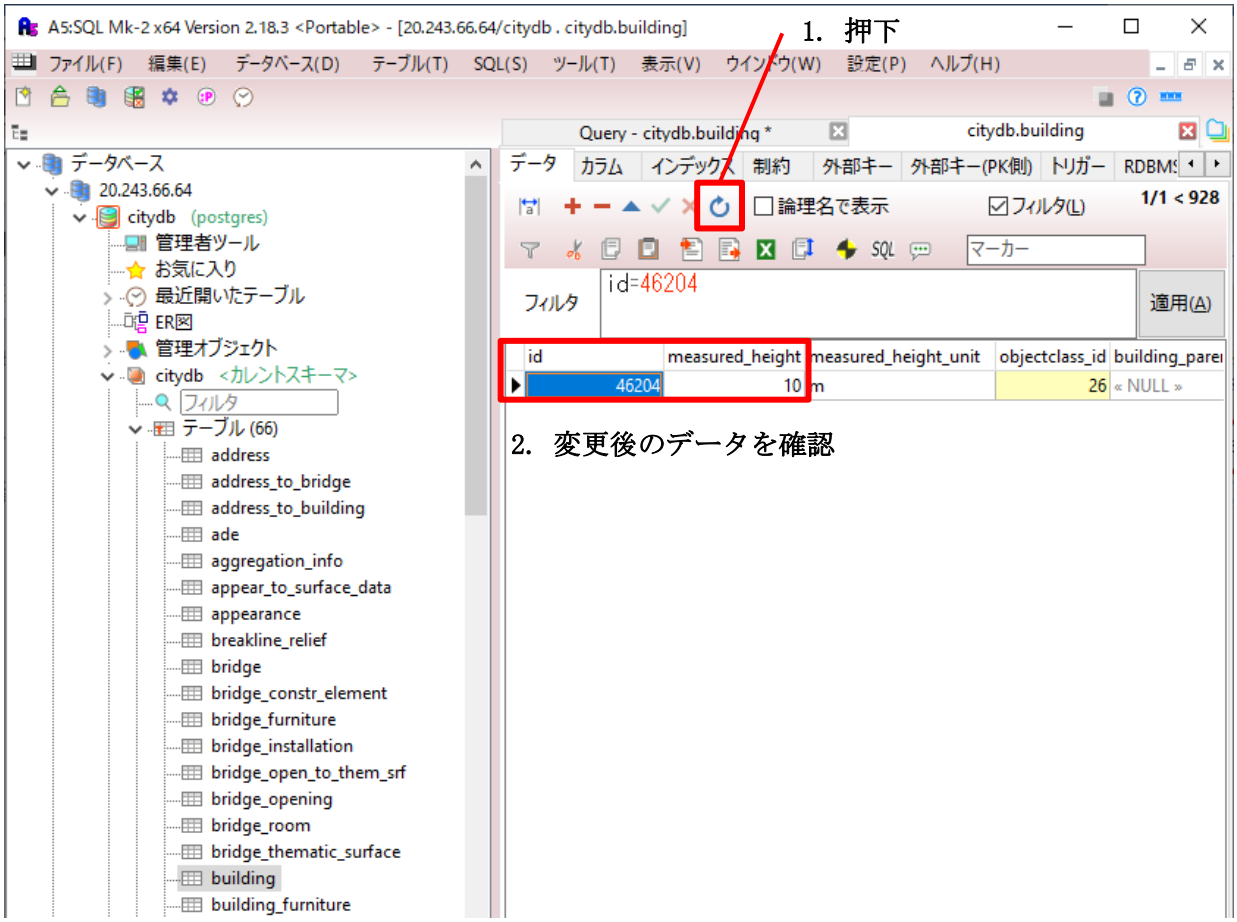


図 3-40 変更後データの確認方法

4. 成果と課題

本レポートでは、3DCityDB で実行できることを確認した。

格納：CityGML 2.0 形式のファイルのインポート/エクスポート（条件による対象の絞り込み）
編集：SQL 文（UPDATE 文）による属性編集
拡張：ADE Manager を用いた XSD ファイルによるデータベース拡張（i-UR 2.0）
データ配信：Web GIS ビューア（Cesium）との連携
エクスポートしたファイル（KML/glTF）を用いての連携となる。（3DTiles 形式での出力はできない。）
3 次元モデルは KML/glTF、地物属性情報はスプレッドシート、又は RESTfull API での連携となる。

3D 都市モデルの管理を CityGML から 3DCityDB に変更した場合、3D 都市モデルの検索と属性更新作業が容易になると考える。CityGML ファイルは基準地域メッシュなどの区画ごとにファイルが存在するため、複数区画においてある条件を満たす 3D 都市モデルを検索する場合は複数の CityGML ファイルに対して検索を行う必要がある。一方、3DCityDB 上に都市モデルデータを格納している場合は SQL の SELECT 文やサードパーティーの DB 管理ツールを使用してデータベース内の 3D 都市モデルを一括で検索可能である。属性更新作業においては、更新対象となる 3D 都市モデルを CityGML ファイル内から探索し、更新対象の属性値を手動で変更するよりも、3DCityDB 上の都市モデルに対して SQL の UPDATE 文を実行して属性値を更新する方が、属性値の誤入力の防止や条件判断による属性値の一括変更などの利便性があると考える。

また、3DCityDB に 3D 都市モデルを格納している場合は Importer/Exporter ツールによって CityGML、CityJSON、KML、COLLADA、glTF ファイルにエクスポート可能である。そのため、変換ツールなどを用意せずに、ユーザーが必要とするファイルフォーマットの 3D 都市モデルを取得することが可能である。

Web GIS ビューア(Cesium)と 3DCityDB との連携では、3D 都市モデルはデータベース内の形状情報を直接参照することは出来ず、KML、glTF ファイルを参照するため、Importer/Exporter ツールによるエクスポート作業が必要となる。今回の調査では未確認であるが、KML 等のタイル構造を持つデータと LOD を持たないデータにおいて都市スケールの大規模データセットを GIS ビューアで描画しようとした場合、LOD を持たないデータが 3DCityDB で管理可能な地物であれば GIS ビューア用にデータをエクスポートする際にタイル構造のデータとして出力可能である。また、3DCityDB のテーブル構造上、LOD を持たないデータであっても LOD1 等のデータとして管理されると予想するため、エクスポート時の visible from 設定を行うことで広域、詳細表示における可視性を制御し、ビューアの描画性能をコントロールできるのではないかと考える。

属性情報に関しては、RESTful API にてビューアが参照する属性値用の公開 API を作成すると、ビューアからデータベース内の属性値を直接参照することが可能である。なお、3D 都市モデルの管理方法を 3DCityDB に変更した場合、Impoter/Exporter のエクスポート機能では 3DTiles ファ

イルをエクスポートできないため、既に PLATEAU で公開している、PLATEAU VIEW（https://plateauview.mlit.go.jp/）で 3D 都市モデルを参照する場合は、現行の CityGML から 3DTiles ファイルへの変換と同様なデータ変換作業が必要となる。

最後に 3DCityDB を導入する場合の課題について記載する。

1 つ目の課題は、3DCityDB へのデータのインポート/エクスポートに関する課題である。Importer/Exporter ツールによって、CityGML 1.0 及び 2.0 に該当するデータはデータベースにインポート又はエクスポート可能であるが、i-UR 2.0 の様な拡張データはデータベースにインポート又はエクスポート出来るように Impoter/Exporter 用のプラグインを作成する必要がある。

2 つ目の課題は、CityGML 3.0 の対応に関する課題である。現行の 3DCityDB は、CityGML 1.0 及び 2.0 に対応しており、CityGML 3.0 には未対応である。3DCityDB の次バージョン（ver.5）にて、CityGML 3.0 の対応を予定しているようであるが、リリース時期は未定である。

本レポートの検討により、3DCityDB が PLATEAU の 3D 都市モデルの管理方法として一定の有用性を持つことが確認できた。具体的には、3DCityDB は PLATEAU のデータ管理に求められる属性管理やデータ変換機能を有している点である。3DCityDB は、属性をデータベースとして管理することができ、特定地物の検索や属性更新の実現など、構造化データである CityGML の特性をいかす機能性を有している。他方、3DCityDB を用いるためには、プラグインの追加（i-UR ADE への対応）や他のソフトウェア（RESTful API）との組み合わせが必要であること、また、データ変換やビューイング機能も有しているが、PLATEAU VIEW 及び CMS に備わっている各機能を包括的に代替するほどの機能性やユーザビリティを有していないことから、3DCityDB 単体での導入は現実的ではない。

以上のことから、PLATEAU VIEW の一部として、属性管理を担う機能として位置付けて活用する方法が適していると考える。

今回の調査では、3DCityDB の構造理解、試験システム環境の構築、i-UR3.0 対応データの導入検証（プラグインの改良等）を行ったが、大容量データの投入による動作確認、データベース性能の確認までは至らなかった。大容量データの取扱いは、PLATEAU においても課題のひとつであり、今回、検証を実施することはできていないが、3DCityDB を活用することで大容量データの検索性能が大幅に向上するため、ユーザーが指定する条件に対応する地物探索性能の改善が期待できる可能性がある。今後は、実運用場面を想定し、広域・大容量のデータを用いた実装検証を行うことが求められる。また、PLATEAU VIEW への 3DCityDB の組み込み手法について、最適なアーキテクチャや技術的な観点で研究を深める必要がある。

5. 用語集

■ 用語の一覧

項目	説明
CityGML （City Geography Markup Language）	地理空間データに関する標準化団体である Open Geospatial Consortium（OGC）が策定した 3D 都市モデルのためのオープンデータモデル及びデータ形式の国際標準。
GML （Geography Markup Language）	Open Geospatial Consortium （OGC）によって開発された地理的な特徴を表現するための XML（Extensible Markup Language）文法。
XML （Extensibule Markup Language）	基本的な構文規則を共通とすることで、任意の用途向けの言語に拡張することを容易としたマークアップ言語の総称。
XSD （XML Schema Definition）ファイル	XML 文書の構造を定義するためのスキーマファイル。
GIS （Geographic Information System）	地理情報システム。地理的位置を手掛かりに、位置に関する情報を待ったデータ（空間データ）を総合的に管理・加工し、視覚的に表示し、高度な分析や迅速な判断を可能にする技術。
Project PLATEAU	国土交通省が進める 3D 都市モデル整備・活用・オープンデータ化プロジェクト。
3DCityDB	空間リレーショナルデータベースの上に仮想 3D 都市モデルを保存、表現、管理するためのジオデータベース。
KML （Keyhole Markup Language）	地理データの表示に使用するファイル形式。 ファイル拡張子は*.kml
COLLADA （COLLAborative Design Activity）	画像、テクスチャ、3D モデルなどのコンテンツを保存可能なファイル形式。ファイル拡張子は*.dae
glTF （GL Transmission Format）	3D モデルを保存するためのファイル形式。拡張子は*.gltf（テキスト形式）、*.glb（バイナリ形式）
Docker	コンテナ仮想化を用いてアプリケーションを開発、配置、実行するためのオープンプラットフォーム。
Docker イメージ	Docker コンテナの動作環境となるテンプレートファイル。Docker イメージには、OS やアプリケー

項目	説明
	ション、アプリケーションの実行に使用するコマンド、メタデータ等を含む。
Docker コンテナ	Docker イメージに基づいてアプリケーションを実行する環境。1 つの Docker イメージを実行すると、１つの Docker コンテナが作成される。
Importer/Exporter	3DCityDB が公開しているデータのインポート/エクスポート機能を提供するツール。
Spreadsheet Genarator Plugin	Importer/Exporter のプラグイン。3DcityDB に保存されている属性データを CSV ファイル、又は Microsoft Excel ファイルに出力する機能を追加する。
ADE （Application Domain Extension）	CityGML の拡張規則。
ADE Manager Plugin	Importer/Exporter のプラグイン。CityGML の ADE を 3DCityDB に登録する機能を追加する。
i-UR	CityGML の拡張規則である ADE に基づいて内閣府地方創生推進事務局が都市再生に必要なデータを拡張した i-都市再生技術仕様（案）。
PostgREST	Web から PostgreSQL を操作する RESTful API を提供するソフトウェア。
3DCityDB-Web-Map-Client	3DCityDB がエクスポートする 3D 都市モデルを閲覧するために開発されたビューアアプリ。
WMS （Web Map Service）	ジオリファレンスが行われた地図画像をインターネット経由で提供するための標準プロトコル。
DTM （Digital Terrain Model）	デジタル地形モデル。
CMS （Content Management System）	コンテンツ管理システム。Project PLATEAU では、PLATEAU 関連データセットを一元管理し、API として公開することを可能とするシステムである。
API （Application Programming Interface）	アプリケーション同士が互いに情報をやりとりする際に使用するインタフェース仕様。

6. 参考資料

■　参考資料の一覧

参考資料	バージョン	参照目的	備考
実証環境構築マニュアル	ver.3.0	PLATEAU の実証環境の調査	
3D 都市モデル標準製品仕様書	ver.3.5	サポート対象の CityGML、i-UR のバージョン	
3dcitydb-docs	最新版（2023/05/16 時点）	3DCityDB の調査	3DCityDB ver.4.4, Importer/Exporter ver.5.3, Spredsheet Genarator Plugin ver.4.2, ADE Manager Plugin ver.2.2 対応
i-UR 2.0 XSD ファイル	-	3DCityDB の i-UR 2.0 拡張検討に使用	
PostgREST Documentation	stable（2023/07/31 時点）	PostgreSQL と PostgREST の連携方法の確認（Tutorial の参照）	

CityGML に最適化されたデータベース構築手法に関する
技術調査レポート

2024 年 3 月　発行
委託者：国土交通省　都市局
受託者：アジア航測株式会社