The New 3D City Database 5.0 - Advancing 3D City Data Management based on CityGML 3.0

Zhihang Yao[1], Claus Nagel[2], Murat Kendir[3], Bruno Willenborg[4], Thomas H. Kolbe[3]

[1] Centre for Geodesy and Geoinformatics, Stuttgart University of Applied Sciences (HFT Stuttgart), Stuttgart, Germany – zhihang.yao@hft-stuttgart.de
[2] Virtual City Systems, Berlin, Germany – cnagel@vc.systems
[3] Chair of Geoinformatics, Technical University of Munich, Germany – (murat.kendir, thomas.kolbe)@tum.de
[4] LIST Eco GmbH & Co. KG, Germany – bruno.willenborg@list-eco.de

Keywords: Geoinformation, Relational Database Modelling, Spatial Database, 3D City Modelling, Digital Twin, CityGML

Abstract

CityGML has become an international standard for semantic 3D city models for over 15 years, and plays a central role in various applications such as urban planning, environmental analysis, and geospatial infrastructure. The recent release of CityGML 3.0 issued by the Open Geospatial Consortium (OGC) introduces significant enhancements to the data model, which offers higher semantic richness and improved interoperability with IoT and BIM domains for urban digital twins. However, these advancements also necessitate the substantial adaptations of many existing CityGML-compliant software systems. One such system is the 3D City Database (3DCityDB), a widely used open-source geodatabase solution for managing 3D city models. This paper presents the new major version 5.0 of 3DCityDB released in early 2025 and redesigned to provide extensive support for CityGML 3.0 while also preserving compatibility with the earlier CityGML versions. The new 3DCityDB v5 introduces a completely reworked relational schema based on a generic mapping principle, which reduces the structural complexity and improves the extensibility significantly. In addition, a novel approach for geometry storage using database-native spatial types also enhances performance and enables seamless integration with third-party GIS platforms. Moreover, a new command-line interface has been developed to support efficient data importing, exporting, querying, and processing workflows. The paper details the underlying system architecture and implementation strategies, and also presents application scenarios and benchmark results. Future research and development plans are outlined as well.

1. Introduction

CityGML has played an important role in advancing the digital representation of urban environments (Kolbe, 2009). It was issued by the Open Geospatial Consortium (OGC) and defines a conceptual model and exchange format for virtual 3D city models. In the past years, its earlier versions 1.0 and 2.0 have been widely adopted worldwide and facilitated a broad range of applications in urban planning, facility management, environmental simulations, and geospatial analysis. With the release of the most recent version 3.0, CityGML now offers a more structured and semantically rich framework to meet the increasing demands of urban digital twins and smart city applications (Kutzner et al., 2020). As a result, the global interest in CityGML 3.0 is rapidly growing. Several pioneering projects and governmental initiatives have already begun adopting the new CityGML version. Notable examples include Munakata City and Taito City in Japan, which released the official CityGML 3.0 datasets under the Japan's Digital Twin Initiative. Other pilot activities and preliminary evaluations were reported in Rotterdam (Netherlands), which considered adopting CityGML 3.0 as the new data basis for its nationwide 3D cadastre system.

Supporting this transition to CityGML 3.0 also poses new technical and conceptual challenges for the 3D City Database (3DCityDB), which is one of the most widely used database solution for managing CityGML-based city models. 3DCityDB is a free and open-source 3D geodatabase solution that has been continuously developed and maintained for nearly 20 years. It is designed to store, manage, and analyse CityGML-compliant models in a standard spatial relational database (Yao et al., 2018). It supports hierarchically structured, semantically rich representations of urban objects across multiple Levels of Detail (LoDs), and is capable of handling very large-scale city models efficiently. Over the past decade, 3DCityDB has been successfully deployed in production systems across numerous major cities around the world such as Berlin, Potsdam, Hamburg, Munich, Frankfurt, Dresden, Rotterdam, Vienna, Helsinki, Singapore, and Zurich. Furthermore, the state mapping agencies of the federal states in Germany store and manage the nation-wide collected 3D city models, including approximately 56 million building models and bridges in CityGML LoD2 using 3DCityDB. However, until the end of 2024, 3DCityDB only supported the earlier CityGML versions 1.0 and 2.0 (Gröger et al. 2012) along with their counterpart CityJSON 1.0 (Ledoux et al., 2019). Since CityGML 3.0 introduces significant modifications and improvements to the data model, a substantial redesign of the 3DCityDB was required to fully support the new CityGML version, as well as to ensure backward compatibility with the earlier CityGML versions.

This paper presents the new 3DCityDB v5 released in March 2025, which provides a full implementation of the CityGML 3.0 Conceptual Model (Kolbe et al., 2021). It introduces several novel concepts and technical innovations to support various encoding formats including GML and CityJSON. For instance, the new 3DCityDB introduces a streamlined and optimized relational schema following more generic mapping principles, which significantly reduce the number of database tables compared to previous versions. It also introduces a new approach for managing CityGML geometries by utilizing database-native spatial data types in a more efficient structure. This enhancement simplifies spatial querying, improves performance, and facilitates direct integration with GIS tools such as QGIS, FME, and ArcGIS. Additionally, a new command-line-based database client has been developed to support the import and export of CityGML 3.0 datasets, perform various data operations, and provide support for the OGC Common Query Language (CQL2). Its lightweight and extendable interface enables efficient integration into automated workflows and complex processing pipelines. The following sections of this paper detail the relevant design decisions, system architecture, as well as the key implementation highlights. We also present real-world use cases and application tests, along with an outlook on future developments and research directions.

2. Related Work and Design Decision

With the release of CityGML 3.0, the existing database solutions for managing 3D city models face a range of new challenges. These typically include the increased semantic complexity and the need for compatibility with different encoding formats. The solutions like CJDB (Powałka et al., 2024) and the earlier 3DCityDB v4 (Yao et al., 2018) were designed with a tight coupling to specific CityGML encoding formats such as JSON and XML/GML, and are therefore not well-suited for adaptation to the conceptual model introduced in CityGML 3.0. In particular, their rigid database schemas usually restrict extensibility and affect performance, especially when handling complex geometric structures or CityGML Application Domain Extensions (ADE) (Biljecki et al., 2018). A notable schema-less alternative is GeoRocket, which was designed for high-performance storage and retrieval of large geospatial datasets using document-based indexing and streaming (Krämer, 2020). While it enables fast data access and processing across various encoding formats, it lacks deep semantic integration, type enforcement, and native support for 3D geometry models. Another alternative has been explored through a Neo4j-based graph database (Son et al., 2017). This solution supports multiple versions of CityGML as well as CityJSON. However, Neo4j does not provide native 3D spatial indexing or spatial query functions. To overcome these limitations, a custom 3D R-tree extension was implemented. While this graph-based approach offers a novel perspective, it remains limited for productive use in terms of spatial performance and native support for complex geometries compared to the modern relational spatial databases such as PostGIS or Oracle.

Unlike the earlier 3DCityDB v4, which followed the traditional GIS data modeling paradigm, where each feature type is mapped to a feature-specific table with predefined attributes, we chose to completely redesign the relational schema for the new 3DCityDB v5. The first motivation for this redesign was that the legacy 3DCityDB schema was tightly coupled to specific CityGML encodings and lacked native support for conceptual extensibility. Extending the legacy schema would have significantly increased its complexity and compromised long-term maintainability. In particular, the number of database tables would have at least doubled, resulting in more complex and slower data queries due to a higher number of performance-critical table joins (Agugiaro et al., 2024). Moreover, integrating the legacy schema with standard GIS tools like FME and QGIS would still have required expert knowledge, mainly due to its fragmented and complex geometry structure (Powałka et al., 2024).

The key research question arising from the redesign is whether a relational database schema based on the Entity-Attribute-Value (EAV) paradigm can effectively support the CityGML 3.0 Conceptual Model, while maintaining performance for real-world 3D city models. The EAV model is known for its open schema design, which allows the system to support arbitrarily extensible feature and attribute definitions without requiring changes to the physical schema. Its generic table structure also makes it adaptable to other data formats such as IndoorGML or ALKIS, and more accessible to standard GIS tools. However, the EAV model has historically been criticized for its potential performance issues in large-scale production systems due to the large size of the central tables and the high complexity of queries. To tackle this challenge, we hypothesized that a hybrid approach, which combines a core EAV structure with a minimal set of dedicated columns or tables for storing certain complex properties such as geometries and appearances, could offer a good balance between semantic richness, extensibility, and efficient processing. To mitigate concerns about performance limitations commonly associated with EAV implementations, we conducted an early test using the large-scale LOD2 city model of Hamburg. The test data were migrated from a 3DCityDB v4 instance to the new database schema using a dedicated SQL migration script. The results showed that, with modern database engines such as PostgreSQL/PostGIS and optimized indexing strategies, the hybrid EAV-based approach preserved the performance of the previous version even for complex queries.

Another key aspect of our research work is that the new database design does not fully adopt the open-schema characteristics of a pure EAV model. Instead, we employed a type-enforced and extensible variation. In this design, all core object types, such as features and data types, are explicitly defined and registered in dedicated catalog tables. Each feature instance and its associated properties reference these type definitions, which allow for consistent type enforcement and schema validation. In addition, this approach also supports flexible extension and reuse of the predefined types, which are particularly effective for supporting CityGML ADEs by registering additional schema definitions aligning with the core schemas.

3. Overview of the New Database Schema

Based on the new EAV-based approach, the database schema has been simplified into 17 tables, which are organized into five logical modules. The relationships between these modules are shown in Figure 1 and described in more detail in the following subsections.

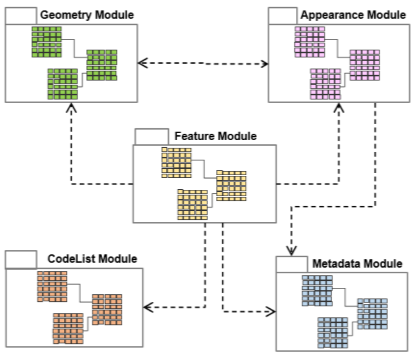


Figure 1. Module Structure of the new 3DCityDB schema.

3.1 Core EAV-Structure

All modules except the Metadata module are mainly responsible for storing city objects along with their attributes, geometries, appearances, and relationships to other features. The EAV-based approach employs a minimal structure consisting of two main tables, namely, *Feature* and *Property* (see Figure 2), which store almost all features and their associated property information without the need for mapping each feature class to a dedicated database table with predefined columns. The *FEATURE* table is the central table serving as the primary storage for all objects such as buildings, roads, or vegetation. Each feature has a unique *id* as its primary key, and an *objectid* string identifier for referencing within databases and datasets. An optional *identifier* column accompanied by an *identifier*_codespace allows distinguishing features across different systems and feature versions globally. The *objectclass_id* indicates the feature type and references the *OBJECTCLASS* table. The bitemporal lifespan information are managed through the *creation_date* and *termination_date*, and *valid_from* and *valid_to* columns. Additional fields like *last_modification_date*, *updating_person*, *reasons_for_update*, and *lineage* provide insights into the origin and update history. Moreover, the spatial *envelope* column stores the minimal 3D bounding box of every feature and can be used for fast spatial querying.

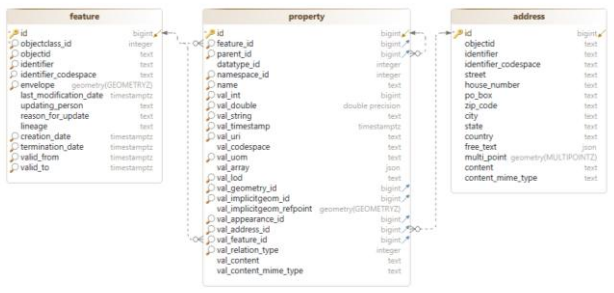


Figure 2. Feature module of the new 3DCityDB schema.

The *PROPERTY* table serves as the central storage location for feature properties. Each property is basically defined by its name, namespace, data type, and value based on a key-value pair design pattern. Based on a type-forced approach, property values are stored in one or more predefined columns that correspond to specific data types allowing for efficient storage and query performance. The data type is determined by the *datatype_id* column pointing to a dedicated metadata table in the metadata module (see Section 3.4). For example, simple primitive types are stored in the columns such as *val_int*, *val_double*, *val_string*, and *val_timestamp*. For non-primitive attributes, array values are stored as JSON structures in the *val_array* column, while arbitrary binary content is stored in val_content with its corresponding MIME type specified in the *val_content_mime_type* column. Complex attributes with nested structures are not stored in a single JSON column. Instead, they are either stored within a single row or represented in a hierarchical manner across multiple rows linked via the *parent_id* column. This design typically enables fine-grained semantic referencing of the individual components within the nested structure.

In addition to storing attribute values, the *PROPERTY* table also represents the semantic relationships between features and other entities. These relationships are maintained in separate rows to ensure clear differentiation between properties and associations. References to other features are stored in the *val_feature_id* column, while geometric associations are represented via the *val_geometry_id* column, which links to the *GEOMETRY_DATA* table (see Section 3.2) and may optionally be qualified by a level of detail via the *val_lod* column. Similarly, implicit geometries are referenced through the *val_implicitgeom_id* column combined with the transformation data stored in the *val_array* and *val_implicitgeom_refpoint* columns. Additional associations include references to appearances and addresses through the *val_appearance_id* and *val_address_id* columns, which link to corresponding entries in the *APPEARANCE* (see Section 3.2) and *ADDRESS* tables respectively. Although address is modelled as a feature type in CityGML conceptual model, it is exceptionally stored in a dedicated *ADDRESS* table rather than in the *FEATURE* table, since this design decision allows for more efficient data querying, indexing, and updates in support of e.g., location-based services.

3.2 Management of Geometry and Appearance

The Geometry module (see Figure 3) includes the tables for storing feature geometries, as well as implicit geometries, which can be reused as templates for multiple features according to the CityGML implicit geometry concept. The *GEOMETRY_DATA* table serves as the central location for storing both explicit and implicit geometry data of the features stored in the database. It supports various geometry types, including points, lines, surfaces, and volume geometries. Explicit feature geometries, which are geometries with real-world coordinates, are stored in the geometry column. This column uses a predefined spatial data type from the database system to represent the geometry data. All explicit geometries must be stored using 3D coordinates in the coordinate reference system defined for the 3DCityDB instance.

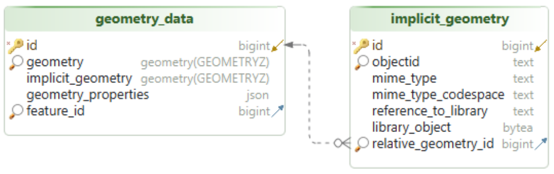
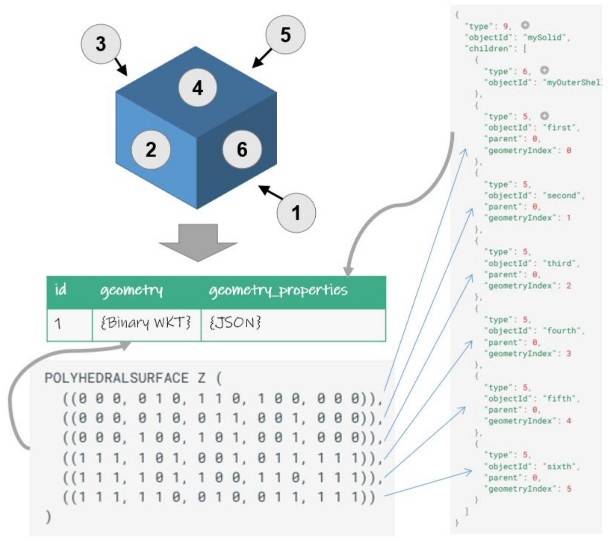


Figure 3. Geometry module of the new 3DCityDB schema.

The use of predefined spatial database types for storing both explicit and implicit geometries present two main challenges. First, CityGML features support a wide variety of geometry types, including primitives such as points, lines, surfaces, and volumes, as well as composite and aggregate geometries, all based on the ISO 19107 spatial schema standard. However, the predefined spatial database types typically cover only a subset of these, and therefore limit the ability to fully represent all CityGML geometries. Second, CityGML allows geometries to be reused by reference and assigned textures or colors. For this, each geometry and its components require unique identifiers. However, spatial database types typically store only raw coordinates and lack the ability to assign unique identifiers, which limits effective referencing and reuse. The previous versions of 3DCityDB addressed these challenges by decomposing surface geometries into individual polygons, each of which is stored in a separate row with a unique identifier. A hierarchical table structure was used to group the polygons according to their original geometry type. While this approach enabled efficient referencing, it required geometries to be recomposed on-the-fly during spatial queries, which made such queries slower and less efficient. Moreover, it increased storage requirements and was limited to surface-based geometries, not points and lines.

The new design of 3DCityDB utilizes a novel approach, which allows using spatial database types to store the entire geometries without decomposition, and can therefore improve spatial query performance and storage efficiency. To preserve the ability to reuse and reference geometries and their parts, and to maintain the expressivity of CityGML geometry types, a JSON-based metadata is stored alongside the geometry in the *geometry_properties* column. To illustrate the structure and use of this JSON metadata, a simple cube solid geometry consisting of six surfaces is used as an example (see Figure 4) to demonstrate, how the approach works in a PostgreSQL/PostGIS database using its specific *POLYHEDRALSURFACE* spatial data type. Unlike the PostGIS polyhedral surface consisting of only a simple array of polygons, the CityGML Solid geometry has an additional level to represent the outer shell formed by the polygons. Besides, the solid, the composite surface, and each polygon have an identifier, that allows the reuse of the component and the assignment of appearance information like textures or colors. The JSON object shown in Figure 4 encodes this extra metadata and links it to the *POLYHEDRALSURFACE* representation.



```
POLYHEDRALSURFACE Z (
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 0 1 0, 0 1 1, 0 0 1, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 1, 1 0 1, 0 0 1, 0 1 1, 1 1 1)),
  ((1 1 1, 1 0 1, 1 0 0, 1 1 0, 1 1 1)),
  ((1 1 1, 1 1 0, 0 1 0, 0 1 1, 1 1 1))
)
```

```
{
  "type": 9,
  "objectId": "mySolid",
  "children": [
    {
      "type": 6,
      "objectId": "myOuterShel
    },
    {
      "type": 5,
      "objectId": "first",
      "parent": 0,
      "geometryIndex": 0
    },
    {
      "type": 5,
      "objectId": "second",
      "parent": 0,
      "geometryIndex": 1
    },
    {
      "type": 5,
      "objectId": "third",
      "parent": 0,
      "geometryIndex": 2
    },
    {
      "type": 5,
      "objectId": "fourth",
      "parent": 0,
      "geometryIndex": 3
    },
    {
      "type": 5,
      "objectId": "fifth",
      "parent": 0,
      "geometryIndex": 4
    },
    {
      "type": 5,
      "objectId": "sixth",
      "parent": 0,
      "geometryIndex": 5
    }
  ]
}
```

Figure 4. Example of storing a 3D solid in geometry data table.

The Appearance module (see Figure 5) enables the storage and assignment of textures and colors to surface geometries. It implements the CityGML appearance concept, where appearances act as containers for surface data, which is mapped to the surface geometries of city objects to define their visual and observable properties. The *APPEARANCE* table is the central component of the appearance module. Each record in the table represents a distinct appearance. Although *Appearance* is also a feature type in CityGML, appearances are not stored in the *FEATURE* table. This is because appearances define the visual and observable properties of surfaces, which are conceptually separate from the spatial features stored in the *FEATURE* table. Each appearance is associated with a specific theme for its surface data, which are used to define the visual appearance of city object geometries, such as textures and material properties, and are managed in a dedicated *SURFACE_DATA* table, that supports different types of surface representations. These include, for instance, surface materials for defining basic properties like color, shininess, and transparency. Texture-related representation can be realized through the so-called parameterized textures, which use texture coordinates or transformations for mapping to the target surface geometries. Georeferenced textures are also supported, which typically rely on real-world spatial references for the surface alignment.

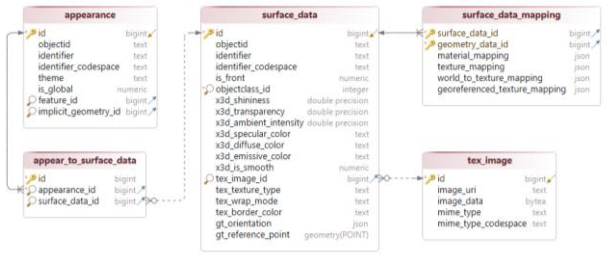


Figure 5. Appearance module of the 3DCityDB schema.

Each surface data is assigned to geometries through a dedicated mapping mechanism that links visual properties such as materials and textures to specific surface elements. These mappings rely on the JSON metadata stored within the *GEOMETRY_DATA* table and are managed via the *SURFACE_DATA_MAPPING* table, which connects surface data entries to their corresponding geometry records. Depending on the type of surface data, a specific mapping is used to for the assignment. For example, materials are assigned by listing the identifiers of the surfaces to which they apply, while textures can be mapped using detailed texture coordinates (see Figure 6) or transformation matrices, that convert real-world positions to texture space. These mappings offer a very flexible way to precisely apply visual properties for complex surface-based geometries.

```
{
  "surface_B": [
    [ [0.0, 0.5], [0.7, 0.3], …, [0.0, 0.5] ], // exterior ring
    [ [0.1, 0.3], [0.6, 0.4], …, [0.1, 0.3] ]  // interior ring
  ],
  "surface_C": [
    [ [0.3, 0.5], [0.1, 0.1], …, [0.3, 0.5] ]  // exterior ring
  ]
}
```

Figure 6. Example of texture mappings for surface geometries.

3.3 Management of Codelists

The Codelist module (see Figure 7) adds support for storing codelists, which are tables of values with the corresponding descriptions or definitions. Many CityGML properties are designed to take values from codelists according to the CityGML 3.0 conceptual model. In practice, codelists may be required, recommended, or suggested by an authority within an organization or community, or more informally defined and used within an application domain.

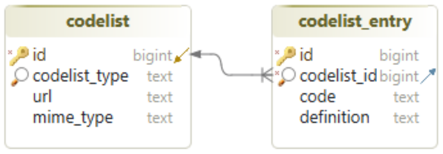


Figure 7. Codelist module of the 3DCityDB schema.

The *CODELIST* table is used to register codelists, each of which is assigned an URL as a unique identifier in database. The *CODELIST_ENTRY* table stores the values of the registered codelists. Each value along with its definition or description is stored in a single table row linked to a codelist. This setup allows for easy lookup of the definition for a code, which is stored as a property value in the *PROPERTY* table, and vice versa. A typical use case is to use the codelist tables to look up or validate property values associated with a codelist during data import and export. Moreover, they can also be used to build a web service, that provides stored codelists as files or serves as a lookup and validation service for individual codelist values.

3.4 Management of Metadata

The Metadata module (see Figure 8) comprises five tables designed to store the meta-information about a 3DCityDB instance and its city model data. First, the *DATABASE_SRS* table stores information about the coordinate reference system defined at setup, which applies to all geometries except implicit geometries. Although the new 3DCityDB is fully based on the CityGML 3.0 standard, it also intentionally supports backward compatibility with CityGML 2.0 and 1.0. This compatibility is a dedicated design of the new 3DCityDB and is achieved by mapping legacy elements to an extended set of predefined types beyond those defined in the CityGML 3.0 conceptual models. A key requirement is that every feature type and attribute must be associated with a namespace. This helps avoid name collisions and logically categorizes the content stored in the database, in order to support multiple versions of CityGML. All namespaces are recorded in the *NAMESPACE* table and populated with a list of 3DCityDB-specific namespaces, which are closely aligned with the thematic modules from CityGML 3.0. The list of namespaces in the *NAMESPACE* table is not exhaustive and can be extended with user-defined namespaces, for example, when registering CityGML ADEs. Each ADE is registered in the *ADE* table as a record assigned a unique identifier along with its meta-attributes e.g., name, version, and description. All ADE classes such as feature and data types are stored in the *OBJECTCLASS* and *DATATYPE* tables respectively, and additionally linked to the corresponding entry in the *ADE* table via the identifier.

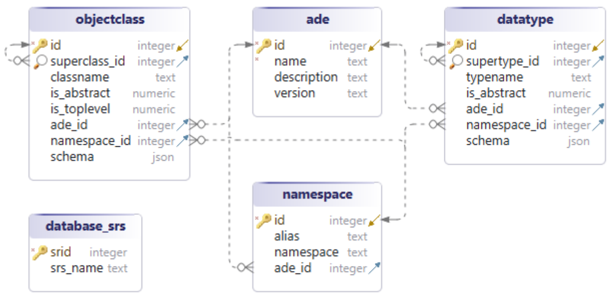
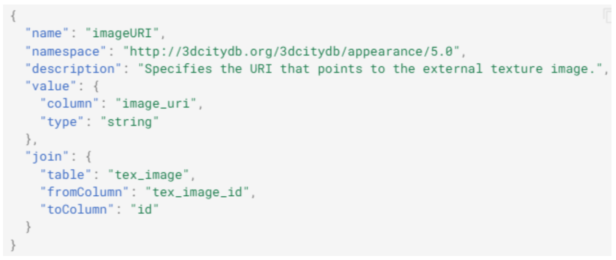


Figure 8. Metadata module of the new 3DCityDB schema.

The *OBJECTCLASS* and *DATATYPE* tables serve as registries for all feature and data types supported by the new 3DCityDB. Each entry is uniquely identified by its name and the associated namespace. Type inheritance is also supported through foreign key references to the corresponding supertypes, which enables reconstruction of the hierarchical structures as defined in the underlying data model. During the database setup, both tables are pre-populated based on the standard CityGML 3.0 conceptual model classes. In addition to type information, both metadata tables contain JSON-based schema mapping that defines, which properties and sub-features a feature or data type may have. This schema mapping serves as the conceptual backbone of the 3DCityDB schema allowing the transition from a purely open schema into a type-enforced flexible schema. Furthermore, the schema mapping also includes the metadata on how to construct joins between database tables for interpreting and querying the stored data (see Figure 9). For example, a software tool can automatically parse and interpret this schema mapping to generate SQL statements from a common query language for interacting with the database. In addition to simple types, the schema mapping also supports complex types, which may include nested properties of simple and complex types. In principle, the database schema is inherently generic and can be extended to support other ISO 19109-based models like IndoorGML, AIXM, ALKIS, or INSPIRE. In such cases, only the corresponding metadata tables and schema mappings need to be adapted.



```
{
  "name": "imageURI",
  "namespace": "http://3dcitydb.org/3dcitydb/appearance/5.0",
  "description": "Specifies the URI that points to the external texture image.",
  "value": {
    "column": "image_uri",
    "type": "string"
  },
  "join": {
    "table": "tex_image",
    "fromColumn": "tex_image_id",
    "toColumn": "id"
  }
}
```

Figure 9. Example snippet of the JSON-based schema mapping.

4. Database Management Tool

As the successor to the “Importer/Exporter” of previous 3DCityDB versions, the new client software called “citydb-tool” is a command-line utility for managing and interacting with the database to store, maintain, and query 3D city models. As with its predecessor, it supports a wide range of operations designed to streamline database workflows allowing for importing, exporting, updating, and deleting 3D model objects. The tool currently supports the latest data formats of CityGML 3.0 and CityJSON 2.0, as well as their earlier versions. In addition, its flexible command-line interface provides a number of general options for configuring logging, loading configuration files, managing plugins, and more. Thanks to its new modular architecture, additional functionalities such as support for other encoding formats can be easily implemented and extended.

4.1 Modular Structure

The new citydb-tool has been completely redesigned based on the Java Platform Module System (JPMS) introduced in Java 9. This modular approach offers improved maintainability and extensibility in terms of its high encapsulation and clear dependencies between components. As illustrated in Figure 10, the component architecture of the citydb-tool is organized into three main categories, namely, *Core*, *User Interface*, and *Utilities* (see Figure 10).

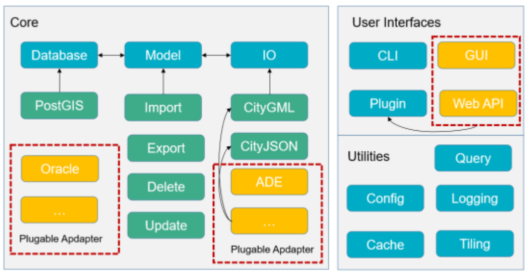


Figure 10. Architecture of the new citydb-tool.

The *Core* components consist of modules for database operations, data model handling, and input/output (IO) operations. A key concept is the definition of a set of abstract APIs, that allow for a common logic to read and write data based on a dedicated intermediate data model, which simplifies the interactions with the database. Concrete implementations of these APIs can be developed and integrated dynamically using pluggable adapters. For instance, an adapter for

PostgreSQL/PostGIS database system has already been implemented. Additional adapters such as one for Oracle or MySQL can also be developed and integrated in the future. The Support for other IO formats in addition to CityGML/CityJSON is also modular. Besides, The *User Interface* category currently already includes a CLI component with various import/export options and also provides a plugin interface allowing developers to implement additional user interfaces such as graphical tools or web-based APIs. The *Utility* category offers a range of helpful APIs for logging, caching, query, data tiling, and configuration management using JSON notation. Moreover, the individual modules are not only intended for internal use within the citydb-tool, but can also be used as embedded libraries for developing custom applications to interact with a 3DCityDB database.

4.2 Main Workflow

The import and export operations supported by the citydb-tool utilize a multi-threaded architecture (see Figure 11) to efficiently process large-scale 3D city model datasets. During data import, the input CityGML or CityJSON files are handled by their respective modular adapters and dispatched to a dedicated Java thread pool for data transformation. In this stage, each top-level feature is concurrently converted as chunks into an intermediate feature representation. The transformed features are temporarily buffered in a waiting queue before being processed by an additional thread pool, which maps and propagates the feature contents into the corresponding database tables. For data export, the citydb-tool first retrieves a buffered ID queue of top-level features either directly from the database or based on a user-defined query. These IDs are then used by a thread pool to fetch the corresponding objects from the database and temporarily store them in a feature queue. In the subsequent stage, a dedicated thread pool converts each feature into the target format using the respective modular I/O adapters, which also handle the serialization of the features into output files in the desired formats. This modular and parallelized workflow enables high-performance import and export operations, facilitating the efficient management of even very large 3D city models using 3DCityDB. Moreover, Docker support has also been introduced to simplify deployment of the new citydb-tool and 3DCityDB to facilitate consistent runtime environments across platforms.

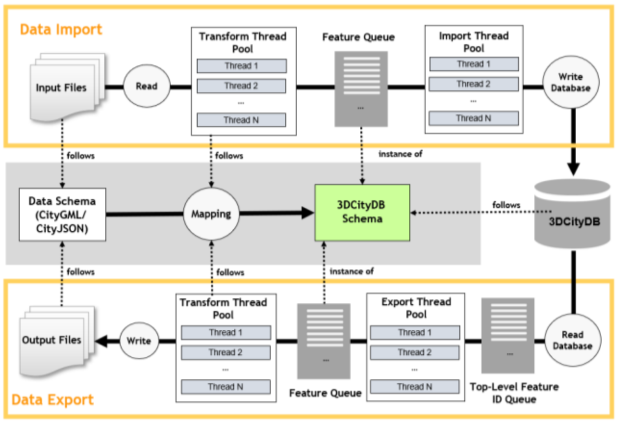


Figure 11. Workflow within the new import/export tool.

4.3 Support for CQL2 Query Language

The Query module (see Figure 10) of the citydb-tool is implemented based on CQL2 (Common Query Language 2), which is an OGC standard offering a generic filter grammar for query options (Vretanos and Portele, 2024). The citydb-tool users can leverage CQL2 to construct precise queries for exporting or deleting features based on attribute values and spatial relationships. For attribute filtering, CQL2 supports the common comparison operators such as *equals* (=), *not equals* (!=), *less than* (<), and *greater than* (>). Spatial filtering further enhances querying capabilities through geospatial functions such as *intersects*, *within*, and *contains*, which determine whether two geometries are overlapped or contained with each other. A typical use case is selecting all city objects within a specified bounding box by evaluating spatial intersections. CQL2 also provides a highly expressive syntax allowing for the construction of complex and nested queries, which use logical operators such as *AND*, *OR*, and *NOT* to combine multiple attribute and spatial filter conditions for refining query results. These queries can be expressed in either plain text or JSON notation, both of which can be interpreted and processed by the citydb-tool. An example of CQL2 query expressed in JSON is shown in Figure 12.

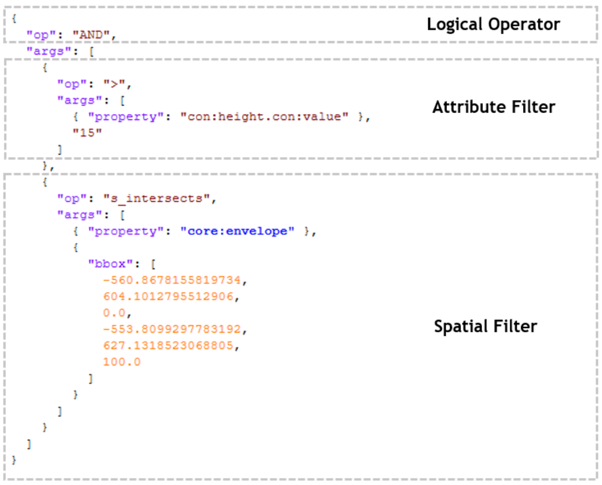

```
{
  "op": "AND",
  "args": [
    {
      "op": ">",
      "args": [
        { "property": "con:height.con:value" },
        "15"
      ]
    },
    {
      "op": "s_intersects",
      "args": [
        { "property": "core:envelope" },
        {
          "bbox": [
            -560.8678155819734,
            604.1012795512906,
            0.0,
            -553.8099297783192,
            627.1318523068805,
            100.0
          ]
        }
      ]
    }
  ]
}
```
Logical Operator
Attribute Filter
Spatial Filter

Figure 12. Example CQL2 query expression in JSON notation.

This CQL2 query example demonstrates a compound filter combining both attribute-based and spatial conditions using a logical operator. While the CQL2 standard is primarily designed for flat and simple features, 3DCityDB operates on a hierarchically structured data model with complex and nested attributes according to the CityGML standard. To support this complexity, we defined and implemented an extended CQL2 that introduces a custom JSONPath-based property notation. It allows for navigation through sub-features and nested attributes based on the schema mapping information stored in the *OBJECTCLASS* and *DATATYPE* tables (see Section 3.4). For example, the property path "*con:height.con:value*" refers to a CityGML construction type and the value of its associated complex property "*height*". The citydb-tool is capable of interpreting such expression and translating it into an optimized SQL statement by leveraging the schema mapping definitions automatically.

5. Evaluation and Testing

The usability of the new 3DCityDB schema has been intensively evaluated and tested regarding the usability with other third-party applications and the import/export processing time.

5.1 Integration with Third-party Software

To conduct the integration test, the open-source software QGIS and the commercial ETL tool FME were selected. The test dataset is a LoD2 building model in CityGML 3.0 format, which includes storeys, boundary surfaces, textures, windows, and doors. It was created by the Institute for Automation and Applied Computer Science (IAI) at the Karlsruhe Institute of Technology (KIT) and is freely available for unrestricted use[1]. The dataset was first imported into a 3DCityDB instance. Three database views were then created to represent different layers for the building's outer shell, the first floor with walls and rooms, and the top floor with roof and loft. These views can be accessed directly in QGIS using its built-in database Manager and displayed as three separate vector layers. Each layer can be visualized and explored not only in 2D, but also in 3D within the QGIS viewer interactively (see Figure 13). QGIS integration has also been investigated by a third party (Tsai et al., 2025), who confirmed the usability of the new 3DCityDB database schema.

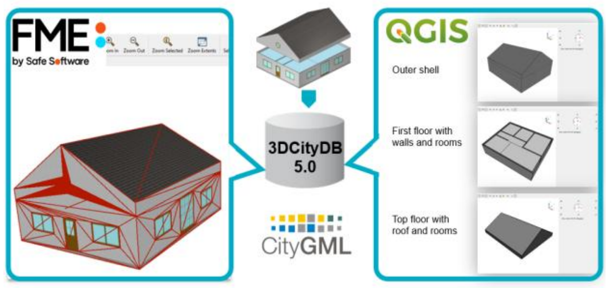
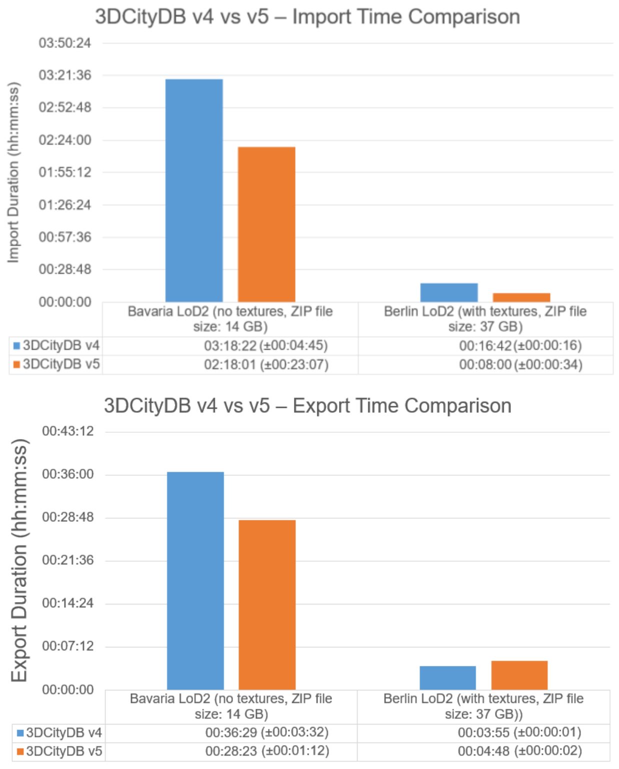

FME
by Safe Software
3DCityDB 5.0
CityGML
QGIS
Outer shell
First floor with walls and rooms
Top floor with roof and rooms

Figure 13. Integration of 3DCityDB with FME and QGIS.

In addition to the QGIS integration, an FME workbench has been developed by the company *Virtual City Systems*, who is one of the main development partners of 3DCityDB. This workbench was implemented based on the FME's database reader to access data from database tables using a customized SQL query. Both geometry and appearance information can be retrieved from the database and visualized together in the FME Viewer (see Figure 13). This also enables seamless conversion into various other formats like 2D/3D Shape, DWG, and OBJ, which are compatible with a wide range of third-party applications. The FME workbench is also designed to be extendable and could serve as an alternative to the aforementioned citydb-tool for interacting with 3DCityDB in future.

5.2 Performance Test

To evaluate the import and export performance of the new 3DCityDB version, a series of tests were conducted using two large open datasets from the federal states of Bavaria and the city of Berlin. The Bavaria dataset contains nearly 9.7 million LoD2 building objects without textures, while the Berlin dataset includes approximately 540,000 LoD2 buildings with full textures. To obtain reliable benchmark results including averages and standard deviations, each test was executed five times on the same Linux server equipped with an Intel® Xeon® w5-2565X processor (3.2 GHz, 18 cores), 128 GB RAM, and 4 TB SSD storage. The system runs Docker on Ubuntu Linux 24.04 LTS. Both the old (v4) and new (v5) versions of 3DCityDB were deployed locally using Docker images running PostgreSQL 17.3 and PostGIS 3.5. As shown in Figure 14, the new 3DCityDB demonstrates improved or at least comparable performance for both data import and export operations. In addition, an export performance test was conducted to evaluate the efficiency of filtered queries in the new database schema. A simple attribute filter was applied to select all buildings higher than 30 meters in the Bavaria dataset. The results showed that 8,369 buildings matched the filter criteria, and the export operation took an average of only 19 seconds. This indicates that, based on new EAV-based hybrid approach, the export performance also scales well when handling subset data from large datasets.



	Bavaria LoD2 (no textures, ZIP file size: 14 GB)	Berlin LoD2 (with textures, ZIP file size: 37 GB)
■ 3DCityDB v4	03:18:22 (±00:04:45)	00:16:42 (±00:00:16)
■ 3DCityDB v5	02:18:01 (±00:23:07)	00:08:00 (±00:00:34)

	Bavaria LoD2 (no textures, ZIP file size: 14 GB)	Berlin LoD2 (with textures, ZIP file size: 37 GB))
■ 3DCityDB v4	00:36:29 (±00:03:32)	00:03:55 (±00:00:01)
■ 3DCityDB v5	00:28:23 (±00:01:12)	00:04:48 (±00:00:02)

Figure 14. Benchmark results of 3DCityDB performance tests.

6. Conclusion and Future Work

In this paper, we present the results of the ongoing development of the new version of 3DCityDB. Compared to its previous versions, 3DCityDB now offers not only improved performance, but also enhanced functionality with extensive support for the latest CityGML 3.0 standard. The redesigned database schema employs an EAV-based and type-forced relational architecture to enhance compatibility with other encoding formats, and also simplifies the implementation of support for CityGML ADEs. Furthermore, the light-weight structure of the database schema improves the overall usability and makes the integration with third-party software such as FME and QGIS easier. The import/export tool has also been reworked based on an improved modular structure, which can be easily extended for additional database systems, data formats, and functional plugins. While the current results represent an important milestone in the development of 3DCityDB, several aspects remain open and will be explored in our future work.

6.1 Full Support for CityGML ADE

Although the new 3DCityDB schema offers a generic and flexible structure for managing CityGML ADEs, there is currently no utility for managing ADE schemas. In fact, the ADE management is simplified compared to earlier versions, as the static EAV-based table structure eliminates the need to

[1] https://www.citygmlwiki.org/index.php?title=FZK_Haus,_CityGML_3.0,_LoD2,_Storeys,_Boundary_Surfaces

create or drop ADE-specific tables. However, to support ADE data import and export, the citydb-tool must be extended, since it lacks a generic implementation for handling arbitrary ADE schemas. While the tool provides a complete API for custom extensions, it still requires in-depth knowledge of the ADE schema. A future research challenge is to investigate whether large language models (LLMs) could be leveraged to automatically generate the necessary code to simplify the ADE integration.

6.2 Support for exporting standardized 3D visualization model

The "VIS-Exporter" plugin in the old 3DCityDB Importer/Exporter allowed users to export the objects in the database as 3D models in KML, COLLADA or gLTF formats. However, a similar plugin must be redesigned based on the new database architecture and to support the now established 3D tile standards such as "i3S" (Belayneh & Reed, 2023) and "3DTiles" (Cozzi & Lilley, 2023), especially for web-based applications. The new geometry storage strategy of the 3DCityDB v5 and the simple structure of the *PROPERTY* table allows us to use the third-party open-source software pg2b3dm to generate 3D tiles with built-in semantic information. Since the main architecture of pg2b3dm is to read and tile geometries stored in *PolyhedralSurface*, *MultiPolygon*, and *MultiLinestring* types within PostGIS databases, it is well-compatible with the new database design. In our experiments so far 3D tiles with different surface materials (but no textures, yet) were created using a thematic set of attributes and displayed in CesiumJS without any additional software. A solution based on an adapted version of pg2b3dm including the selection of different levels-of-detail, styling of object types, and extraction of thematic data will be released soon.

6.3 Support for OGC API - Features

The previous versions of 3DCityDB supported the Web Feature Service (WFS) for web-based access to 3D city objects. In the new 3DCityDB version, WFS is no longer supported and is intended to be replaced by the modern "OGC API - Features", which is also an OGC standard for creating, querying, and modifying spatial data on the web (Portele et al., 2022). Unlike WFS, the new API follows RESTful principles and improves the interoperability with web applications. It typically delivers data in GeoJSON formats allowing seamless integration with GIS tools. In the future, 3DCityDB is expected to support this API with additional formats like CityGML and CityJSON. Since the API uses CQL2 for data filtering, the existing citydb-tool already provides a solid foundation for the implementation. Moreover, the CQL2 extensions developed in the context of 3DCityDB v5 could be proposed to OGC for consideration in future versions of the CQL2 standard.

References

Agugiaro, G., Pantelios, K., León-Sánchez, C., Yao, Z., Nagel, C., 2024. Introducing the 3DCityDB-Tools Plug-In for QGIS. *Recent Advances in 3D Geoinformation Science*. Lecture Notes in Geoinformation and Cartography. Springer Nature Switzerland, Cham, 797–821.

Biljecki, F., Kumar, K., Nagel, C., 2018. CityGML Application Domain Extension (ADE): Overview of Developments. *Open Geospatial Data, Software and Standards* 3(1), 13.

Belayneh, T., Reed, C., 2023. OGC Indexed 3d Scene Layer (I3S) and Scene Layer Package (*.slpk) Format Community Standard Version 1.3, Open Geospatial Consortium.

Cozzi, P., Lilley, S., 2023. 3D Tiles Specification Version 1.1, Open Geospatial Consortium.

Gröger, G., Kolbe, T. H., Nagel, C., Häfele, K.-H., 2012. OGC City Geography Markup Language CityGML Encoding Standard version 2.0, Open Geospatial Consortium.

Krämer, M., 2020. GeoRocket: A scalable and cloud-based data store for big geospatial files. *SoftwareX* 11, 100409.

Kolbe, T. H., 2009. Representing and Exchanging 3D City Models with CityGML. In: Lee, J., Zlatanova, S. (Eds.), *3D Geo-Information Sciences*. Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg, Berlin, Heidelberg, 15–31.

Kolbe, T. H., Kutzner, T., Smyth, C. S., Nagel, C., Roensdorf, C., Heazel, C., 2021. OGC City Geography Markup Language (CityGML) Part 1: Conceptual Model Standard v3.0, Open geospatial consortium.

Kutzner, T., Chaturvedi, K., Kolbe, T. H., 2020. CityGML 3.0: New Functions Open Up New Applications. *PFG - Journal of Photogrammetry, Remote Sensing and Geoinformation Science* 88(1), 43–61.

Ledoux, H., Arroyo Ohori, K., Kumar, K., Dukai, B., Labetski, A., Vitalis, S., 2019. CityJSON: A compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*, 4(1), 1–12.

Nguyen, S. H., Yao, Z., Kolbe, T. H., 2017. Spatio-semantic comparison of large 3D city models in CityGML using a graph database. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-4/W5, 99–106.

Powałka, L., Poon, C., Xia, Y., Meines, S., Yan, L., Cai, Y., Stavropoulou, G., Dukai, B., Ledoux, H., 2024. *CJDB: A Simple, Fast, and Lean Database Solution for the CityGML Data Model*. Recent Advances in 3D Geoinformation Science. Lecture Notes in Geoinformation and Cartography. Springer Nature Switzerland, Cham, pp. 781–796.

Portele, C., Vretanos, P., Heazel, C., 2022. OGC API - Features - Part 1: Core corrigendum, Open Geospatial Consortium.

Tsai, B., Agugiaro, G., Leon-Sanchez, C., Nagel, C., Yao, Z., 2025. Introducing server-side support for 3DCityDB 5.0 to the 3DCityDB-Tools plug-in for QGIS. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* (same volume as this paper).

Vretanos, P., Portele, C., 2024. Common Query Language (CQL2), Open Geospatial Consortium.

Yao, Z., Nagel, C., Kunde, F., Hudra, G., Willkomm, P., Donaubauer, A., Adolphi, T., Kolbe, T. H., 2018. 3DCityDB - A 3D Geodatabase Solution for the Management, Analysis, and Visualization of Semantic 3D City Models Based on CityGML. *Open Geospatial Data, Software and Standards* 3(5), 1–26.