Extending 3DCityDB with Oracle AI Database Support and LLM-Based Natural Language Interfaces

Zhihang Yao[1], Karin Patenge[2], Huiling Gong[3], Claus Nagel[4], Thomas H. Kolbe[5]

[1] Centre for Geodesy and Geoinformatics, Stuttgart Technical University of Applied Sciences (HFT Stuttgart), Stuttgart, Germany – zhihang.yao@hft-stuttgart.de
[2] Oracle Global Services Germany GmbH – karin.patenge@oracle.com
[3] Oracle Corporation, USA – huiling.gong@oracle.com
[4] Virtual City Systems, Berlin, Germany – cnagel@vc.systems
[5] Chair of Geoinformatics, Technical University of Munich, Germany – thomas.kolbe@tum.de

Keywords: Geoinformation, Urban Digital Twin, Large Language Model, Spatial Database, Oracle AI, 3DCityDB, CityGML

Abstract

The open-source 3D City Database (3DCityDB) is a widely used solution for storing and managing semantic 3D city models based on the CityGML standard. Its latest version 5 introduces a redesigned and highly generic schema along with a JSON-based schema mapping mechanism, which was initially supported only by PostgreSQL with the PostGIS extension. This paper presents the extension of 3DCityDB to the Oracle AI Database, which offers a database-native natural language interface for interacting with the database. Following a model-driven approach, we ported the complete schema from PostgreSQL to Oracle within a unified relational modelling environment. We implemented a lightweight Oracle adapter that shares a common interface with the PostgreSQL version in the *citydb-tool* to keep import, export, and query workflows consistent across both database platforms. One of the main research challenges arises when integrating LLM-based natural-language-to-SQL (NL2SQL) through the Oracle SELECT AI package. It typically conveys database semantics via static annotations on fixed physical tables and columns, whereas 3DCityDB v5 adopts a generic Entity-Attribute-Value (EAV) schema whose semantics are encapsulated as JSON in metadata tables. As a result, annotations alone cannot express the structure required for correct query generation. To bridge this gap, we propose a schema-driven approach that automatically derives a dedicated database view from the JSON schema mappings and renders it into a compact, LLM-readable context, which is then injected into the SELECT AI prompt. This enables general-purpose LLMs to resolve hierarchical relationships and generate recursive SQL.

1. Introduction

Over the past 20 years, the open-source 3D City Database (3DCityDB) has been widely utilized for the efficient storage and management of CityGML data, serving as the foundation for Urban Digital Twins (Yao et al., 2018). Recently, 3DCityDB version 5 was released along with a completely redesigned architecture, offering a highly generic and flexible schema that provides improved performance while supporting various data models, including the latest CityGML 3.0 standard (Kutzner et al., 2020). It also incorporates a novel schema mapping mechanism that describes the semantic structure of the database, which enables the programmatic interpretation of the semantic database structure based on a JSON-structured mapping description (Yao et al., 2025).

However, the new version of 3DCityDB initially supported only the open-source PostgreSQL database with the PostGIS extension and lacked support for enterprise solutions such as the Oracle Database. Since the latest version of Oracle Database, now branded Oracle AI Database, already offers numerous optimizations and native AI features allowing applications to access and process data more efficiently, porting the PostgreSQL implementation to Oracle has become a high priority. This transition mainly involves three implementation and research challenges. First, to remain consistent with the PostgreSQL version, a model-driven approach is required to generate the 3DCityDB schema without manually hard-coding SQL scripts. Second, a common interface that is compatible with both Oracle and PostgreSQL must be established to allow applications seamless access to and interaction with both database types. Finally, the 3DCityDB must be fine-tuned or extended to leverage Oracle's AI capabilities for natural-language-to-SQL (NL2SQL) querying against the 3DCityDB schema.

In this paper, we first present a model-driven approach for representing relational databases using a unified modelling platform compatible with both Oracle and PostgreSQL. Using this method, all 3DCityDB tables were ported by mapping individual data types, constraints, and both normal and spatial indexes from the original PostgreSQL version. Once the schema is defined, SQL creation scripts are automatically derived from the database model. For database interaction, including import, export, and delete functions, we implemented an Oracle database adapter, which shares the same interface as the PostgreSQL/PostGIS version and hence ensures that I/O workflows and query capabilities remain consistent across database platforms.

A significant highlight of this research is the integration of Large Language Models (LLMs) with 3DCityDB for the automatic translation of natural language into SQL within Oracle databases. To facilitate complex attribute, spatial, and hierarchical queries over semantic 3D city models, we proposed a schema-driven approach that introduces a new database view called PROPERTY_CATALOG into the 3DCityDB schema. This view is generated automatically from the 3DCityDB JSON

schema mappings without the need for any manual maintenance. It employs a flattened tabular structure to store meta-information, including inheritance hierarchies, relational dependencies, and the complex attribute structures of all feature types. By explicitly representing storage columns, aggregation relationships, and navigable paths, the property catalog view can provide the essential context for a system prompt, which facilitates LLMs in generating hierarchical SQL queries, including those involving complex and recursive join statements.

2. Database Implementation

To implement the 3DCityDB database for Oracle using a model-driven architecture, we utilized DbSchema, which is a powerful visual environment software for collaborative design and schema documentation (DbSchema Development Team, 2026). By leveraging this tool, both the PostgreSQL and Oracle versions of 3DCityDB were implemented based on a unified relational modelling approach, which significantly simplifies long-term maintenance and cross-platform synchronization. As shown in Figure 1, all database tables along with their related constraints and indexes for every 3DCityDB logical module were fully mapped from the PostgreSQL version to their Oracle counterparts. Furthermore, a complete SQL script file for establishing the database schema can also be automatically generated from the relational database diagram.

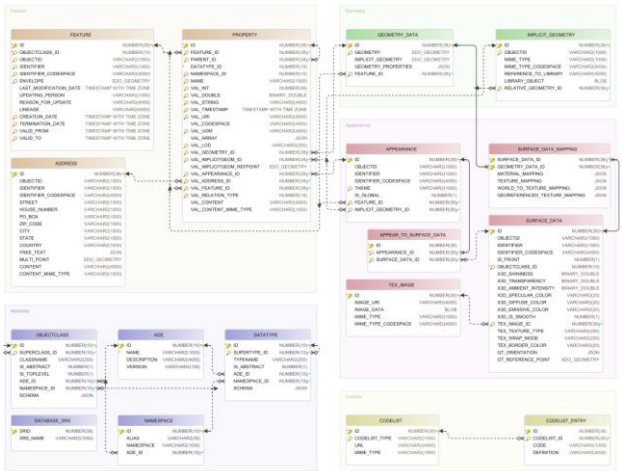

Figure 1. 3DCityDB relational diagram for Oracle.

The data management of 3D city models is realized by extending the *citydb-tool*, which is a standard command-line application bundled with the 3DCityDB package. This tool follows a modular architecture organized around a common set of abstract classes and interfaces, which separates the database-independent application logic from database-specific implementation details and thereby supports the consistent integration of different database systems (Yao et al., 2025). Since this abstraction layer exposes a uniform interface, the logic shared by all database-specific modules is implemented only once, and the appropriate database adapter can be discovered and activated automatically at runtime. As shown on the left-hand side of Figure 2, a common core module defines the abstract adapter classes, such as *SchemaAdapter* and *GeometryAdapter*, together with the helper interfaces for query operations, spatial operations, and temporary data. Building on this design, the Oracle database adapter remains highly lightweight, since it only needs to extend the abstract base classes and implement the helper interfaces, as shown on the right-hand side of Figure 2. Its responsibilities are therefore limited to mapping data types, converting between the geometry model and Oracle's native spatial representation, and translating spatial queries into the corresponding Oracle Spatial operations. By encapsulating these database-specific implementations while inheriting all remaining functionality from the common core, the Oracle adapter can be integrated seamlessly with the *citydb-tool* and provides full support for querying and processing CityGML data.

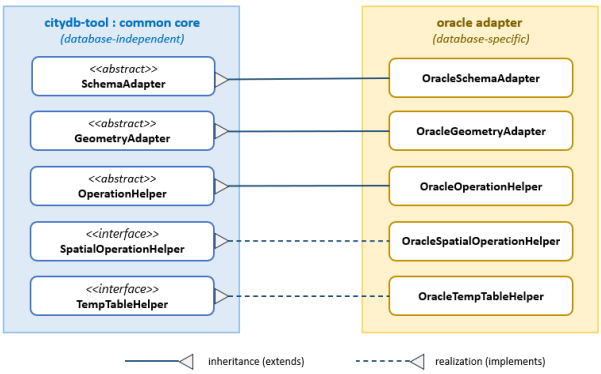

Figure 2. *citydb-tool* adapter architecture for Oracle database.

3. Extending 3DCityDB with AI Support

The 3DCityDB is a fully semantically enriched relational database that stores not only the physical geospatial data but also the semantic data in dedicated tables. The logical schema mappings of the object model onto the relational model are organized as well-structured JSON objects, which expose the database structure in a machine-readable form. This provides a solid foundation for programmatically constructing suitable prompts for Large Language Models (LLMs), enabling them to interpret that structure and interact with the database through natural language. The core of this capability lies in a key technology that translates natural language into structured SQL statements by means of the Oracle Database's AI capabilities. In this research work, the 3DCityDB has been extended accordingly, and the results are presented in detail in the following subsections.

3.1 NL2SQL for 3DCityDB

Translating natural language questions into executable SQL (NL2SQL, or Text-to-SQL) has long stood at the intersection of natural language processing and database management, aiming to let users interact with databases more easily. Earlier research relied mainly on deep learning or purpose-built models, which typically require a dedicated model to be trained specifically for the NL2SQL task (Katsogiannis-Meimarakis & Koutrika, 2023). Recently, the rapid development of LLMs has redefined the state of the art. Popular general-purpose LLMs such as Claude, Gemini, and the GPT family now exhibit strong zero- and few-shot NL2SQL ability through in-context learning, without the need for any task-specific fine-tuning (Hong et al., 2025; Sobo et al., 2024). As a result, the central challenge developers face is no longer training a dedicated model, but rather how to present the database schema and its semantics to a general-purpose LLM through effective prompt engineering to satisfy user queries (Gao et al., 2024).

Enabling natural-language interaction with a geospatial database such as the 3DCityDB raises exactly this schema-presentation challenge. A recent agent-based framework tackles it effectively by encoding the database structure and providing the schema to an LLM through a retrieval mechanism, which enables

NL2SQL translation across diverse stakeholder queries (Kanna & Kolbe, 2025). This design places the schema knowledge and the NL2SQL logic in a dedicated application layer above the database, which offers considerable flexibility in orchestrating multiple tools and data sources. A complementary design point worth exploring is where this translation capability resides. When the schema knowledge and NL2SQL logic live in a standalone agent service, they are naturally coupled to that application, and other clients accessing the database directly do not share in these capabilities. In addition, the service needs to be deployed and maintained separately, and the query text is passed to an external model for processing. These are reasonable trade-offs for an application-centric architecture. An alternative and orthogonal direction is to mirror the conventional SQL-based interaction by embedding the NL2SQL capability directly within the database. This way, the client can send natural-language text to the database directly, and the database performs the translation internally and returns the response directly to the user. This keeps the translation within the database boundary and makes it accessible to any client without an intermediary service.

3.2 Integration of Oracle Select AI

The latest Oracle AI Database 26ai comes with a so-called "Select AI" feature, which integrates NL2SQL directly into the database and enables direct interaction with a range of different LLMs (Oracle, 2026). It therefore provides seamless translation of natural-language user queries into SQL and their subsequent execution within the database. To realize this, SELECT AI relies on annotations that serve as semantic metadata attached to tables and columns. This metadata is automatically assembled into a well-structured prompt and sent to the LLM, enabling it to reason about the underlying database schema. Additional user-defined context can be attached to the prompt along with the query and passed to the LLM for generating results. In addition, Oracle SELECT AI offers a rich set of modes (actions) for customizing the response. For example, *SELECT AI showprompt* returns the generated prompt in the response, which is especially useful for developers to debug the table and column annotations as well as any user-defined context. *SELECT AI showsql* previews the SQL generated by the underlying LLM without executing it against the database and is hence helpful for reviewing a query before it is run. Finally, *SELECT AI runsql* is the default action, executes the generated query directly and serves as the standard interface for client applications.

To enable 3DCityDB to leverage Oracle Select AI, a conventional approach is to add annotations or comments to the individual data tables, such as FEATURE, PROPERTY, GEOMETRY_DATA, etc. However, the architecture of the new 3DCityDB v5 presents a unique challenge for this standard implementation. Whereas conventional relational models store each attribute in a dedicated, fixed column, 3DCityDB v5 adopts a generic relational schema based on a generic mapping principle, which is conceptually like an Entity-Attribute-Value (EAV) model (Dinu & Nadkarni, 2007). In this schema, property names are not fixed columns. They are rather stored as rows, and the corresponding schema mappings are encapsulated in JSON within separate metadata tables such as DATATYPE and OBJECTCLASS. Consequently, annotating physical database columns alone is insufficient, as their names do not convey the underlying semantic definitions required for interpretation (Shi et al., 2025). The schema mapping information must therefore also be attached to the prompt sent to the LLM. However, Oracle SELECT AI cannot read the data records in the tables when building the prompt. To address this, a database-side procedure is implemented to read and parse the JSON schema mappings and generate LLM-friendly prompt text. This text is then appended to the system prompt generated by Oracle Select AI. Using this approach, the LLM can autonomously interpret the semantic model structure of the 3DCityDB schema and generate SQL statements in response to user's natural language queries. The technical implementation takes the form of an Oracle Database package in which the entire pipeline is written in SQL and consists of three main steps (see Figure 3).

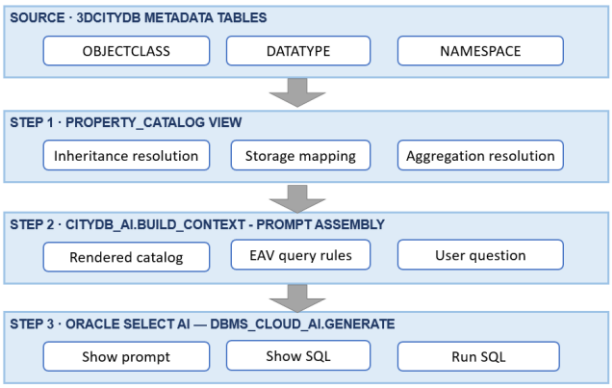

Figure 3. The three-step in-database pipeline that resolves 3DCityDB v5 schema-mapping metadata into an augmented prompt for Oracle Select AI.

As a first step, we introduce a database view called PROPERTY_CATALOG, whose contents are automatically derived from the 3DCityDB metadata tables, and which consolidates the schema-mapping information into a single queryable structure. To construct the view, the pipeline first scans the OBJECTCLASS metadata table and resolves inheritance by traversing the class hierarchy from each concrete feature class to its root ancestor. It then collects the inherited property definitions from the JSON Schemas of the ancestor classes and assembles a resolved set of properties for each feature class. Each property is annotated with the ancestor class in which it is declared. The pipeline also scans the DATATYPE metadata table to identify the storage column or sequence of table joins required to access each property. Complex data types containing nested attributes are decomposed into individual sub-attributes, each of which is resolved independently. For every sub-attribute, the pipeline records both the parent property name and the corresponding storage column. This decomposition ensures that even deeply nested structures remain fully queryable within the flattened representation. Finally, the pipeline resolves aggregation references between feature classes. If a referenced feature class is abstract and therefore cannot be directly instantiated, the pipeline recursively expands its subclasses and maps the reference to the complete set of concrete feature classes that may occur in its place according to the data model (Kolbe et al., 2021).

In the second step, once the PROPERTY_CATALOG is built, a dedicated database procedure named *build_context* renders it into a compact text prompt (see Figure 4). For properties inherited from a common ancestor, and thus shared by many concrete feature classes, the procedure emits each one only once and groups them into a named block keyed by that ancestor. Each concrete feature class is then rendered as a short header that references the blocks it inherits and lists only the properties it declares itself, so that no inherited property is repeated across feature classes. The pipeline also appends a reverse aggregation

https://doi.org/10.5194/isprs-archives-L-4-W2-2026-253-2026

index that maps each child feature class to the aggregation properties that reference it, allowing target-first pathfinding across the aggregation hierarchy. This rendered catalog, together with the EAV query rules and the user question, is subsequently injected into the prompt of the native Oracle SELECT AI call. We route the prompt through this intermediate catalog view rather than rendering it directly from the raw schema-mapping metadata, so that the costly resolution logic, like recursive inheritance traversal and abstract-to-concrete expansion, is expressed only once and cached in the database. The catalog view also exposes a flat relational structure that decouples the renderer from the nested JSON metadata encoding and can be independently inspected, validated, and reused by other consumers such as debugging tools or downstream applications.

```
== INHERITED PROPERTY BLOCKS ==
@core:AbstractFeature
  creationDate → feature.creation_date
  ...
... more blocks (@core:AbstractCityObject, @con:AbstractConstruction, ...)

== FEATURE TYPES ==
[bldg:Building id=901 TOPLEVEL] : @core:AbstractFeature @con:AbstractConstruction
  class → property.val_string
  boundary → contains WallSurface(709),RoofSurface(712), ... via VAL_FEATURE_ID
  ...
... more feature types

== CONTAINED-BY INDEX ==
  WallSurface(709) ← boundary [con:AbstractConstruction]
  RoofSurface(712) ← boundary [con:AbstractConstruction]
  ... one line per concrete boundary-surface type
```

Figure 4. Excerpt of the compact text prompt rendered from the PROPERTY_CATALOG view.

In the final step, the assembled prompt is processed according to the selected action. As noted at the beginning of this section, the prompt can also be returned without invoking the LLM. This mode enables rapid inspection of the assembled prompt and is therefore useful for debugging. When invoking the LLM, users can either retrieve the generated SQL statement or execute it directly against the database to obtain the query results. All three calls follow the same call pattern and differ only in the value assigned to a single parameter. The following listing illustrates this workflow. The *build_context* function first combines the user's question - "*How many windows and doors are contained in the building?*" - with the rendered catalog to form a single prompt. The *profile_name* parameter specifies the LLM profile to be used, whereas the *action* parameter selects one of the three operating modes: returning the assembled prompt (*showprompt*), returning the generated SQL statement (*showsql*), or executing the SQL statement and returning the results (*runsql*). DBMS_CLOUD_AI.GENERATE is the entry function that applies all these settings and sends the request to the LLM. The overall call takes the form of a regular SELECT statement and can therefore be integrated easily into database client applications.

```
SELECT
  DBMS_CLOUD_AI.GENERATE(
    prompt =>
      citydb_ai.build_context(
        'How many windows and doors are
contained in the building'),
    profile_name => 'OPENAI',
    action       => 'showsql'
  ) AS result
FROM dual;
```

Listing 1. Example SELECT AI call returning the generated SQL (*showsql* action).

4. Evaluation and Testing

To evaluate the presented development and research results, we conducted two groups of tests. The first group consists of import and export tests against the database schema implemented for Oracle, which benchmark the overall performance of both the database itself and the Oracle adapter that we developed for the *citydb-tool*. The second group evaluates our approach to extending the NL2SQL capabilities of Oracle Select AI, in which the correctness of the SQL generated by the LLM from the injected prompt is assessed. The system architecture that underlies both groups of tests is summarized in Figure 5.

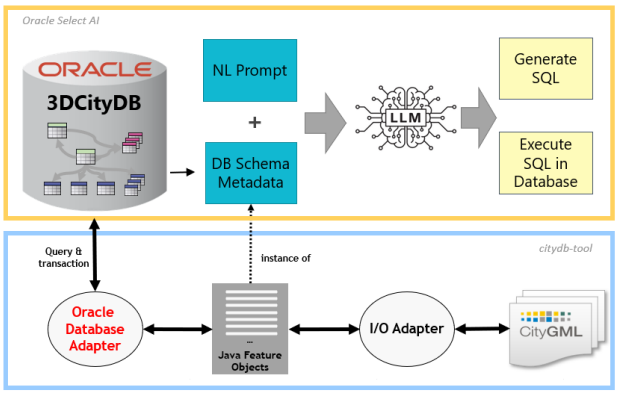

Figure 5. Architecture and workflow for Oracle integration with SELECT AI capabilities.

4.1 Performance Test of Import and Export

4.1.1 General remarks

To evaluate the import and export performance of the 3DCityDB tool using the new adapter written for the Oracle AI Database 26ai, a series of benchmark tests was carried out using two publicly available LoD2 city model datasets of substantially different size (see Table 1).

	Parameters	Berlin	Bavaria
Import	Total size (Bytes)	16,593,445,921	303,844,160,558
	# Files	926	36,595
	# Buildings	641,813	10,102,802
Full export (.gz)	Size (Bytes)	2,094,212,177	1,713,039,137
	# Files	1	1
	# Buildings	641,813	10,102,802
Filtered export (.gz)	Size (Bytes)	6,975,481	94,786,568
	# Files	1	1
	# Buildings	1,686	8,664

Table 1. LoD2 CityGML Datasets

Various settings of table definitions and database session parameters were applied to test their influence on the import and export performance. After each test, the data tables were emptied, and sequences to generate unique identifiers were reset. The 3DCityDB changelog was disabled for the import tests to reduce additional write overhead.

4.1.2 The environment

The environment was deployed as a client/server architecture to match typical production environments.

The database server was provisioned as Oracle Base Database Service (DB system version: 23.26.2.0.0) using an AMD VM.Standard.E5.Flex shape (Oracle Enterprise Linux 9.7, 16 OCPU, 256 GB memory, 16 Gbps network bandwidth, theoretical max IOPS of 256K) and with additional block volume attached (16 TB data storage, 4 TB recovery area storage).

The client machine containing the installed 3DCityDB software stack was provisioned on a compute VM (Oracle Enterprise Linux 9.7, 1 OCPU, 1 Gbps network bandwidth, 12 GB memory) with two additional block volumes attached (1 TB and 8 TB Higher Performance, iSCSI, 20 Volume Performance Units).

The Higher Performance option provides a linear performance scale of 75 IOPS/GB up to a maximum of 50,000 IOPS per volume. Throughput scales at the rate 600 KB/s/GB up to a maximum of 680 MB/s per volume.

Both datasets were stored on the client machine. Compared to an environment, where both the client and database server are deployed on the same physical hardware, the network latency has an impact on the performance. Considering the number and size of the files to be transferred to the database and a sustained throughput of 1 Gbps, the transfer time can be estimated using the formula:

time_in_sec = total_size_in_bytes / sustained_bytes_per_sec (1)

where:

sustained_bytes_per_sec for 1 Gbps = 125,000,000

For the Berlin dataset, data transfer took approximately 2 minutes and 13 seconds, whereas for the substantially larger Bavaria dataset, it took approximately 45 minutes.

4.1.3 Test results

The *citydb-tool*, including the Oracle adapter, was configured with the best-performing options (see Table 2).

Options	Setting
Changelog	DISABLED
No of threads	16
Log level	INFO
Index mode	drop
initSQL	PARALLEL DML/DDL/QUERY, PARALLEL_DEGREE_POLICY=AUTO

Table 2. Import/export configuration

Across all tests, the largest performance improvements in index creation and data export were achieved by using partitioned tables, local indexes, and up-to-date optimizer statistics (see Table 3). The performance results presented in Table 4 and Table 5 are based on four test runs conducted using the best-performing import and export options and database settings.

Database features	Setting
Partitioning	HASH partitioning
Indexes	LOCAL
Compression	NOCOMPRESS
Logging	NOLOGGING
Degree of parallelism	32
Sequence cache	100,000
Tablespace	Big file
Statistics	Gathered after import and before export
DB_WRITER_PROCESSES	4

Table 3. Best-performing Oracle AI Database configuration

Run	Import	Index creation	Full export	Filtered export
1	00:16:29	00:00:57	00:20:10	00:00:22
2	00:16:36	00:00:57	00:19:54	00:00:13
3	00:16:30	00:00:50	00:19:54	00:00:14
4	00:16:28	00:00:49	00:19:49	00:00:13

Table 4. Execution times – Berlin dataset

Run	Import	Index creation	Full export	Filtered export
1	03:06:13	00:16:08	03:18:50	00:01:34
2	03:07:15	00:16:22	03:17:29	00:01:08
3	03:06:22	00:16:24	03:18:25	00:01:09
4	03:06:08	00:16:22	03:17:18	00:01:09

Table 5. Execution times – Bavaria dataset

4.1.4 Test summary

The repeated benchmark runs indicate that the Oracle AI Database configuration delivers stable performance for large-scale 3DCityDB import and export workflows. Variations in import and full-export runtimes were minimal, suggesting that the measured results are reproducible rather than isolated best-case observations.

The most significant performance improvement was the substantial reduction in index creation and filtered export times using partitioned tables, local indexes, and up-to-date optimizer statistics.

The results indicate favorable scalability from the Berlin to the Bavaria dataset and show that selective exports remain efficient even for very large city models.

Overall, the benchmark results demonstrate that the Oracle adapter enables 3DCityDB to support high-volume operational data management workflows and achieves performance consistent with the objectives reported for 3DCityDB v5 (Yao et al., 2025).

4.2 Test of NL2SQL using Oracle Select AI

To test and evaluate the NL2SQL approach developed for the Oracle-based implementation of 3DCityDB, we selected a complex 3D building model of the Stuttgart University of Applied Sciences (HFT Stuttgart). The model includes a detailed indoor structure and a rich hierarchy of rooms, storeys, windows, and doors, organized through multiple levels of containment that form a well-defined aggregation hierarchy.

Figure 6 illustrates both this aggregation structure and a visualization of the model.

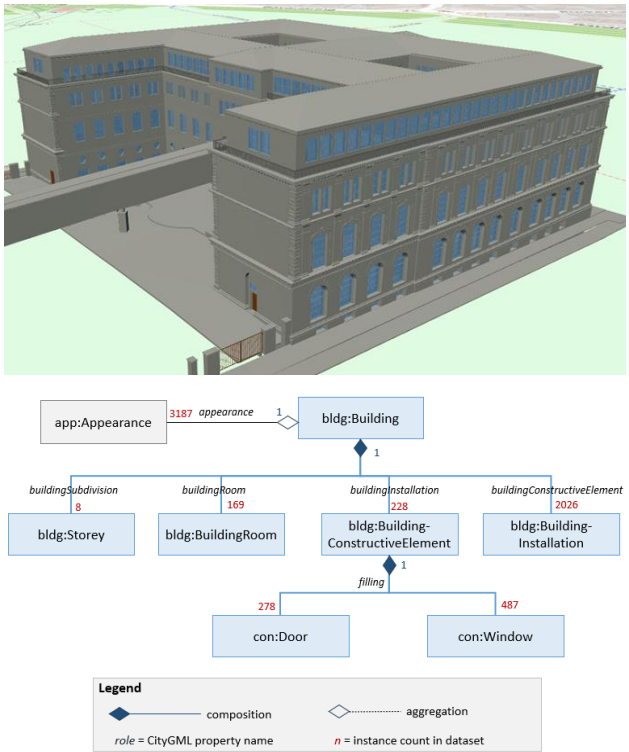

Figure 6. 3D visualization and aggregation hierarchy of the HFT 3D building model, including the instance counts per feature class.

The HFT Stuttgart building model originated as a Building Information Model (BIM) in Industry Foundation Classes (IFC) format (Laakso & Kiviniemi, 2012). For the NL2SQL evaluation, it was therefore converted to CityGML 3.0 using the open-source *ifc-to-citygml3* tool developed by TUM (Kolbe, 2026). The tool maps IFC entities to their corresponding CityGML feature types; for example, an *IfcDoor* entity is mapped to a CityGML *Door* feature.

The evaluation was conducted using the Docker image of Oracle AI Database 26ai Free (Release 23.26.2.0.0). Although SELECT AI is included in Oracle AI Database 26ai, it is not pre-configured in the Free container image. We therefore extended the 3DCityDB installation scripts to configure SELECT AI automatically after the Oracle-based 3DCityDB instance has been created. Once the CityGML data had been prepared and the database configured, the dataset was imported into the Oracle database using the Oracle adapter for *citydb-tool* developed as part of this work. The example natural-language query presented in Listing 1 was then executed using the OpenAI model GPT-5.4. The resulting SQL statement is shown in Listing 2. As illustrated in the listing, the model generates a hierarchical query that follows the 3DCityDB schema structure encoded in the injected prompt. The query counts of *Window* and *Door* features exactly match those shown in Figure 6, supporting the correctness of the generated SQL statement.

The same natural-language query was also evaluated using the Gemini 3.5 Flash and Claude Opus 4.8 models. Both models generated structurally similar SQL statements and yielded the same correct results.

```
WITH "sub" ("id") AS (
  SELECT
    "f"."ID" AS "id"
  FROM "CITYDB"."FEATURE" "f"
  WHERE "f"."OBJECTCLASS_ID" = 901
  UNION ALL
  SELECT
    "p"."VAL_FEATURE_ID" AS "id"
  FROM "sub" "s"
  JOIN "CITYDB"."PROPERTY" "p"
    ON "p"."FEATURE_ID" = "s"."id"
   AND "p"."VAL_RELATION_TYPE" = 1
   AND "p"."VAL_FEATURE_ID" IS NOT NULL
)
SELECT
  COUNT(
    DISTINCT CASE
      WHEN "f"."OBJECTCLASS_ID" = 704
      THEN "f"."ID"
    END
  ) AS "window_count",
  COUNT(
    DISTINCT CASE
      WHEN "f"."OBJECTCLASS_ID" = 705
      THEN "f"."ID"
    END
  ) AS "door_count"
FROM "CITYDB"."FEATURE" "f"
JOIN "sub" "s"
  ON "s"."id" = "f"."ID"
WHERE "f"."OBJECTCLASS_ID" IN (704, 705)
```

Listing 2. SQL statement automatically generated by GPT-5.4 from the prompt assembled by the CITYDB_AI package and Oracle Select AI.

As shown in Listing 2, GPT-5.4 correctly identifies that aggregation relationships in the 3DCityDB schema are stored as rows in the PROPERTY table. It generates a recursive common table expression (CTE) that begins with the building feature (OBJECTCLASS_ID = 901) and traverses the aggregation hierarchy through VAL_FEATURE_ID, which references the nested child features. The filter condition VAL_RELATION_TYPE = 1 restricts the traversal to aggregation relationships, whereas a value of 0 indicates a regular association. This recursive approach collects all descendant features, irrespective of their nesting depth. The query then counts the *Window* (OBJECTCLASS_ID = 704) and *Door* (OBJECTCLASS_ID = 705) features among the descendants.

In addition to this representative query example, we evaluated more than 20 natural-language queries covering several categories, including simple and complex attribute queries, spatial queries, and recursive aggregation queries spanning multiple hierarchy levels. The generated SQL statements were manually validated against reference results obtained directly from the database and produced correct results in all evaluated cases. As a baseline comparison, SELECT AI did not generate correct queries for the 3DCityDB schema when the PROPERTY_CATALOG was omitted.

5. Conclusion

In this paper, we presented the extension of the 3DCityDB to the Oracle AI Database, together with its enrichment by LLM-based NL interfaces. Following a model-driven architecture, the complete 3DCityDB schema was ported from PostgreSQL to Oracle within a unified relational modelling environment, from which the creation scripts for both platforms are generated and maintained. A lightweight Oracle adapter was integrated into

the *citydb-tool* through the same abstract interfaces used by the PostgreSQL version, so that data import, export, and query proceed consistently regardless of the underlying database system.

The main research contribution of this work is a schema-driven approach that brings natural-language querying directly into the database by combining Oracle SELECT AI with the generic, JSON-based schema mappings of 3DCityDB. Rather than annotating the fixed physical tables and columns, which is incompatible with the EAV-style design of the schema, we automatically derive a PROPERTY_CATALOG view that flattens inheritance hierarchies, storage mappings, and aggregation relationships into a simple and queryable form. From this view, a compact prompt text can be generated that allows LLMs to autonomously produce the SQL statements corresponding to user queries, without task-specific fine-tuning or an additional external application layer. The evaluation with a complex CityGML building model confirmed that queries generated by several state-of-the-art LLMs returned correct results.

Future work will first focus on systematic benchmarking of import and export performance for the Oracle database, including the potential gains from Oracle's parallelization and spatial indexing capabilities. In addition, a broader evaluation of NL2SQL accuracy will be conducted across more diverse and complex queries and against more real-world datasets. We also plan to extend the approach to further LLMs, ranging from public commercial models to lightweight open-source ones such as Alibaba's Qwen, so that the framework can be deployed in a local environment. Finally, since the presented approach is inherently generic by purely reading the metadata tables, we will investigate whether CityGML Application Domain Extensions (ADEs), such as the Energy ADE (Agugiaro & Padsala, 2025) and Urban Planning ADE (Ishimaru et al., 2020), can also be supported automatically without additional manual effort.

References

Agugiaro, G., Padsala, R., 2025. A proposal to update and enhance the CityGML Energy Application Domain Extension. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* X-4/W6-2025, 1-8. https://doi.org/10.5194/isprs-annals-X-4-W6-2025-1-2025.

DbSchema Development Team, 2026. DBSchema Online Documentation, Version 10.3.0. https://dbschema.com/documentation/ (30 June 2026).

Dinu, V., Nadkarni, P., 2007. Guidelines for the effective use of entity–attribute–value modeling for biomedical databases. *International Journal of Medical Informatics* 76(11-12), 769-779. https://doi.org/10.1016/j.ijmedinf.2006.09.023.

Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., Zhou, J., 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proceedings of the VLDB Endowment* 17(5), 1132-1145. https://doi.org/10.14778/3641204.3641221.

Hong, Z., Yuan, Z., Zhang, Q., Chen, H., Dong, J., Huang, F., Huang, X., 2025. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. *IEEE Transactions on Knowledge and Data Engineering* 37(12), 7328-7345. https://doi.org/10.1109/TKDE.2025.3609486.

Ishimaru, N., Kurokawa, C., Tanaka, Y., Oishi, T., Akahoshi, K., Kutzner, T., Kolbe, T. H., 2020. CityGML Urban Planning ADE for i-Urban Revitalization (OGC Discussion Paper 20-000r1). Open Geospatial Consortium. https://docs.ogc.org/dp/20-000r1/20-000r1.pdf.

Kanna, K., Kolbe, T. H., 2025. Advanced User Interaction with Urban Digital Twins using Large Language Models. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences X-G-2025,* 469-476. https://doi.org/10.5194/isprs-annals-X-G-2025-469-2025.

Katsogiannis-Meimarakis, G., Koutrika, G., 2023. A survey on deep learning approaches for text-to-SQL. *The VLDB Journal* 32(4), 905-936. https://doi.org/10.1007/s00778-022-00776-8.

Kolbe, T. H., 2026. IFC to CityGML 3.0 Converter, Technical University of Munich. https://github.com/tum-gis/ifc-to-citygml3 (30 June 2026).

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. https://doi.org/10.1007/s41064-020-00095-z.

Laakso, M., Kiviniemi, A., 2012. The IFC Standard: A Review of History, Development, and Standardization. *Journal of Information Technology in Construction* 17, 134-161. https://api.semanticscholar.org/CorpusID:14218203.

Oracle, 2026. Oracle AI Database SELECT AI User's Guide, Release 26ai (G35918-06). Oracle Corporation. https://docs.oracle.com/en/database/oracle/oracle-database/26/selai/oracle-database-select-ai-users-guide.pdf (30 June 2026).

Shi, L., Tang, Z., Zhang, N., Zhang, X., Yang, Z., 2025. A Survey on Employing Large Language Models for Text-to-SQL Tasks. *ACM Computing Surveys* 58(2), Article 54. https://doi.org/10.1145/3737873.

Sobo, A., Mubarak, A., Baimagambetov, A., Polatidis, N., 2024. Evaluating LLMs for Code Generation in HRI: A Comparative Study of ChatGPT, Gemini, and Claude. *Applied Artificial Intelligence* 39(1), Article 2439610. https://doi.org/10.1080/08839514.2024.2439610.

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. https://doi.org/10.1186/s40965-018-0046-7.

Yao, Z., Nagel, C., Kendir, M., Willenborg, B., Kolbe, T. H., 2025. The New 3D City Database 5.0 - Advancing 3D City Data Management based on CityGML 3.0. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences X-4/W6-2025*, 241-248. https://doi.org/10.5194/isprs-annals-X-4-W6-2025-241-2025.