The study of Building Information Model server focusing on IFC to RDB conversion

Zhenglu ZHU¹, Kazuya SHIDE²

¹ Ph.D. Candidate, Graduate School of Eng, Shibaura Institute of Technology
² Prof, School of Architecture, Shibaura Institute of Technology

Abstract

This study proposes that the Building Information Model should aim to become a data service provider in order to expand the possibilities of utilizing BIM data. In order to construct a BIM server, this study devised a design method for creating a database of IFC, an international standard specification. We have developed a program to automatically create an RDB from IFC data using various mapping rules. Furthermore, benchmark tests were conducted to compare the performance difference and behavior of each DB to clarify the appropriate RDB design method for storing IFC data. As a result, we believe that the method of creating a table for each entity and maintaining the inheritance relationship by giving it a foreign key is the most appropriate method for constructing an IFC server at present. As BIM continues to develop, there is a growing momentum to share and utilize BIM data on cloud servers. We will contribute to the realization of such an ideal.

Keywords

Building Information Model, buildingSMART International, IFC Server, Relational Database, SQL, Auto-mapping

1. Background

In recent years, the emergence of new technologies, such as IOT, big data, AI and robotics, has created a series of innovative services and businesses by sharing various types of knowledge and information. The Japanese government is currently striving to achieve a human-centered society (Society 5.0) that balances economic development and solutions to social issues through systems that highly integrate virtual and real space, including smart cities [1].

The Ministry of Land, Infrastructure, Transport and Tourism has launched the so-called “Project Plateau” initiative (hereinafter referred to as “Plateau”) that reflects this approach [2]. 3D city models are being developed to reproduce real-world cities within a virtual world. A 3D urban model is an urban spatial information system that reproduces the urban space itself digitally by labeling objects within urban spaces, including buildings and streets, with facility information such as the name, use and year of construction. Meanwhile, Building Information Model (BIM) systems are becoming increasingly popular in the building industry, with 3D urban models and BIMs each playing different roles. While 3D city models provide geospatial information from a macroscopic perspective, BIM can provide detailed information for one or more buildings from a microscopic level. Accordingly, combining the two will pave the way for advanced urban design, environmental simulation and more, which will help refine urban development.

Plateau, which was developed based on CityGML, converted BIM models in different data formats into Industry Foundation Classes (IFC), an international standard BIM data specification and further mapped IFC elements (Entity) to CityGML elements (Class), then tried to seamlessly transfer data between the two systems via a reconversion method. However, as things stand, only a small portion of Entity from IFC can be strictly mapped to the CityGML,
which, in turn, means only fractional BIM data can be handled by the Plateau server on the end-user side. Accordingly, while there is no particular problem when only the indoor spatial information of architecture is used, utilizing more detailed architectural data for use in 3D city models is a challenge [3]. Accordingly and to expand the scope of BIM data usage, this study proposes that BIM data should not be contained in a 3D urban model as a mere information source, but should aim to become a data services provider.

This goal is difficult to achieve for the following possible reasons. Firstly, BIM does not merely represent the target building in the form of 3D objects, but also holds various building-related information, including materials, performance and costs, comprising a huge amount of wide-ranging data. The model also stores a huge amount of dynamic data generated during the operational building phase. Secondly, the BIM data handled by each practitioner (architect, designer, construction company, etc.) during each phase (design, construction, operation, etc.) taps into a range of BIM software according to the respective purpose of use. This means that even within a single project, many BIM models emerge; each with different data specifications. Furthermore, recent years have seen software manufacturers develop a whole new range of new BIM software packages, one after the other. More and more data leads to the diversity of data specifications becoming increasingly complex and makes it difficult to integrate BIM data and link information with diverse industries.

2. Research objectives

In this study, we will develop a method for building a BIM server that is usable independently of BIM software to collaborate with BIM data of all kinds and in bulk. Using a BIM server is not only key to integrating complicated BIM data generated during design, construction and maintenance phases, it also allows users to roll out services that link integrated BIM data with information via a Web Application Programming Interface (API).

Constructing a BIM server depends on having the specification of the BIM data to be processed and a means of storing it permanently. Meanwhile, the sheer range of BIM software available means the BIM model concept can be defined and constructed with multiple data schemas. The IFC developed by buildingSMART International (bsi, formerly known as IAI) is intended to ensure interchangeability among BIM models by defining an open format data schema. In addition, the use of data linkage by IFC in practice has been active and the effectiveness of using IFC as a standard specification for BIM data has been recognized.

Since a BIM model holds all kinds of information about a building, it handles data in bulk. Currently, the mainstream approach to housing data in volumes of this size is so-called file-based data integration, which involves creating the original BIM data in each software package, then exporting it to an IFC file for further integration. However, building lifecycle information integrated into a file not only causes the file size to increase exponentially, the user of the data must also be aware of the physical structure of the IFC file when programming to extract the necessary data. However, the extremely complex hierarchical structure of IFC and the difficulty in deciphering its contents makes this impractical. Moreover, maintaining data consistency is difficult when multiple users share and use a single file. Conversely, there are three advantages to utilizing a database: data association, data sharing with simultaneous access and blocking unauthorized data or backing up. Accordingly and to solve the above-mentioned problems, IFC data can be integrated, managed and shared more effectively by constructing an IFC-based database, which is sometimes also referred to as an IFC model server. The main objective of this study involves presenting a design method of a database appropriate for storing IFC data and clarifying any problems associated with the same.

3. Literature review

By now, many researchers have been working on the realization of the IFC model server. Databases can be broadly classified into two types: relational (SQL) databases (hereinafter referred to as RDB) and non-relational (NoSQL) databases. In the study of IFC’s DB, databases are divided into two types, SQL and NoSQL, depending on the type of data base management system (DBMS) adopted.

For NoSQL databases, Watanuki devised a method of storing data models that is independent of IFC schema version differences using a graph database [4], [5]. Beetz et al. developed a BIMserver using a key-value store DBMS [6]; NoSQL, unlike RDB. The fact that it is "schema-less" and not fixed to a table structure means even large volumes of IFC data can be processed rapidly, although frequent data updates, deletions and other processes may render the data inconsistent.

Adachi reproduced the IFC schema defined in the EXPRESS language in the highly versatile
eXtensible Markup Language (XML) language in accordance with the ISO 10303-28:2007 standard for RDBs [7]. Furthermore, the XML-based IFC schema (IfcXML) was converted to SQL Server, an RDBMS provided by Microsoft and an IFC model server called IMSvr was developed. Unfortunately, the IMSvr server is currently no longer maintained and the conversion mechanism is not yet publicly available. In the meanwhile, YOU developed the CIS2 SQL server [8]. Like IFC, it was defined in the EXPRESS language, so it can be substituted as an IFC server. However, CIS/2 has a special relationship called ANDOR between each entity, which impedes system development compared to when an IFC-based server is used. Another method GUO involved converting an IFC schema to Java classes and then MySQL using Hibernate ORM (Object-Relational Mapper) [9]. The common factor between these studies is that each researcher devised mapping rules for converting the IFC schema to RDB and showed the potential to convert IFC to RDB. However, it remains unclear how the various mapping rules affect the data integrity, QUERY performance for retrieving data and other performance aspects of the constructed DB. Moreover, given that the IFC server described above is unavailable as open source, it is not possible for a third party to directly use and compare it.

The main drawback of using RDBs to store IFC data is the difference in data definition philosophies on both sides: IFC is defined in an object-oriented manner, unlike RDBs [10]. Accordingly, with RDBs, the trade-off between table normalization and performance is unavoidable. In response, LEE and Li showed that an Object-Oriented, object-relational database (OODB) is more efficient than and outperforms RDB [11], [12]. However, OODB technology is immature, its interoperability among various OODBMS is low and APIs for connecting DBs and applications are not unified, hampering any practical implementation. Conversely, since RDBMS include many tools and functions with interoperability, porting data between different systems is simple.

4. Research Methodology

Currently, since the original BIM files are not created on an IFC basis, bidirectionally linking IFC files exported from different BIM software may involve "thinning out" the IFC files into a standardized data structure and checking the validity of the converted IFC files. This prevents duplicate, inconsistent, or missing data when adding, deleting, or updating data. In addition, the structured query language (SQL) can be used to process complex data cases accurately. Given that RDBs have a long history and are technologically mature and reliable, this study considers RDB an appropriate method for storing IFC server data permanently and attempts to convert IFC data structures into RDB. Specifically, we will devise all mapping rules for converting IFC to RDB and develop a program to automatically create RDB from IFC schema files in line with various mapping rules. Furthermore, by conducting benchmark tests, we will compare the performance difference and behavior of each DB and clarify the appropriate RDB design method for storing IFC data.

5. The structure of the IFC schema

The IFC schema is written in EXPRESS, the Formal Data Specification Language and the same applies to the "other" category. The properties of each entity are expressed as attributes and the concrete values assigned to them are called instances. Each attribute can be a SimpleType (simple data type), a NamedType (named data type, which includes EntityType and UserDefinedType), an AggregationType (aggregation data type), an instance of an entity, or an instance of an object. ConstructedType (a constructed data type, of which SelectType (selection type) and EnumType (enumeration type) are included). Attributes are also classified into EXPLICIT (positive attributes), DERIVE (induced attributes) and INVERSE (reverse attributes), with only positive attribute data described in the IFC physical file (.ifc). Although basic modeling can be done by assigning attributes to entities, EXPRESS introduces the concept of an inheritance relationship to enable more advanced modeling. The attributes defined in an entity are inherited from an entity of a lower type through inheritance. Accordingly, the essence of converting IFC to RDB is the challenge of how to reproduce IFC schema components such as entities, explicit attributes and inheritance relationships between data types and entities in an RDB specification. Each figure must be accompanied by a single caption, to appear beneath, and cited in the text. Figures should appear in the order in which they are first mentioned in the text and numbering of figures must continue through any appendices. Similarly, each table must be accompanied by a single caption, to appear above the table, numbered and mentioned in the text. Figures and tables should be placed to the top or bottom of the area.
6. Different types of mapping rules

6.1. Entity

The basic table design method for mapping the IFC schema to RDB can be divided into two major types: "entity-to-table" (entities mapped to tables) and "attribute-to-table" (attributes mapped to tables), as shown in the figure 1. The former includes three more possible patterns depending on the method used to store the inheritance relationship between entities. The advantage of is that data can be retrieved quickly because all information held by each subentity can be retrieved without using JOIN SQL, which joins multiple tables. In addition, an identification column must be added to the table to recognize a particular subentity. However, this approach must allow NULL values in columns other than the primary key column and the identification column. For example, the IFC schema defines a required attribute called IsMilestone for IfcTask, a subentity of IfcProcess. Conversely, IfcProcedure, another subentity of IfcProcess, has no IsMilestone attribute defined in the first place, which means that IsMilestone can be regarded as a non-mandatory attribute for IfcProcedure. Consequently, data integrity cannot be ensured on the DB side because the IsMilestone column is forced to allow a NULL value to be able to store data in a single table. In addition, IFC has a multi-level inheritance relationship. For example, there are 419 sub-entities derived from IfcRoot, with 602 attributes. Accordingly, the number of columns in a single table tends to be excessive, degrading DB performance. Accordingly, the P1 method was deemed unsuitable for DBing IFC and excluded from consideration in this study. In P2, a table is created for each entity and the table of a subentity contains columns for its own attributes and columns for its inheritance relationship with a super-entity. In addition, foreign key constraints are set for the columns of the inheritance relationship and the primary key of the super-entity is preserved. For example, when you store an instance of one IfcTask in the DB, you store only the values that correspond to the Status, WorkMethod, IsMilestone, Priority, TaskTime and PredefinedType attributes you have declared in the T_IfcTask table. The other attribute values are registered in the table of each super-entity. In addition, by sharing the primary key, each table becomes associated with the others, allowing specific information to be retrieved using JOIN SQL. This can be seen as both an advantage and a disadvantage, because creating a table for each entity means many tables have to be joined when retrieving data across the entity hierarchy, which reduces performance. Unlike the P1 method, however, P2 allows the IsMilestone column to have a NOT NULL constraint, thus maintaining data integrity. Accordingly, the number of tables in the entire DB can be reduced and it is thought to outperform P2 because data can be retrieved across the entity hierarchy without using JOIN SQL. However, with this approach, when multiple sub-entities inherit one and the same super-entity, there are many duplicate columns between tables, which wastes DB space. Moreover, when an entity has a relationship with another super-entity, the relationship cannot be expressed in the DB because no super-entity table is created. For example, according to the IFC scheme, an IfcSweptDiskSolid (a solid created by sweeping a cross-section) has a one-to-one relationship with an abstract-type entity, IfcCurve, via the Directrix attribute, which is a super-entity transferable from IfcCurve to IfcLine, Ifc. In the P3 method, the table must have information about which table is tied to IfcSweptDiskSolid. This makes data storage possible, but since foreign keys cannot be set between tables, the data state of the related tables cannot be guaranteed and scope to maintain integrity depends completely on the application side. In addition, since the IFC schema itself was developed while being updated sequentially, in the unlikely event that an abstract entity is updated, the DB built accordingly must be drastically restructured. In contrast, in the P2 method, if an abstract entity is updated, only the table of the corresponding entity must be modified. (1) This maximizes the reduction in the number of DB tables and makes the data easier to manage. However, since attribute information is stored in a distributed manner, when retrieving information on an entity, each related attribute table must be combined and searched, which may hamper data retrieval.

6.2. SimpleType, EnumType, UserDefinedType

Simple data types are the most basic of all data types in EXPRESS and include REAL, INTEGER, NUMBER, LOGICAL, BOOLEAN, STRING, BINARY (binary type). As shown in the table 1, if a type corresponding to SQLserver exists, it is converted as is, but given the lack of any data type corresponding to SQLserver for logical and Boolean types, the bit type is used as a substitute. All enumerated types in IFC can be represented as character types, so the value of an attribute defined with an enumerated type is converted to 1 if TRUE, 0 if FALSE and NULL if UNKNOWN. For enumerated types in IFC, the value of an attribute defined in
an enumerated type converts to narchar because it can be represented as a character type. For user-defined types, it depends on the underlying data type.

### 6.3. AggregationType

To begin with, the relational data model does not explicitly support arrays as a data type, so it is not possible to directly represent the aggregate data type in EXPRESS. To make this possible,
two workarounds are suggested: one involves combining the elements of an aggregate data type by separating them with commas and storing them as character types. The other involves creating a separate table in which to store each element. In the former, not only is the first normalization violated, if data that should be represented by each data type is uniformly converted to a character type or if the IFC file to be stored in the DB contains incorrect data, the DB side cannot detect the incorrect data and the incorrect IFC data is stored as is. To avoid this, the program must ascertain the data validity before importing the data. In addition, when retrieving data stored in the DB, the program must reverse the conversion to the original data type, which makes the system development more complex. The latter, conversely, regards the attributes of an entity and the aggregate data type it has as a one-to-many relationship and creates a table for each. Furthermore, providing foreign keys paves the way to guarantee consistency between related tables in the DB. For example, two IfcCartesianPoints with coordinates (X=0, Y=0, Z=0) and (X=0, Y=0, Z=100) are stored in the DB. The figure 2 shows that, in terms of the field IfcCartesianPoint_Id, one record in the left table is associated with several in the right table at the same time. If a foreign key constraint is set on this field and an attempt is made to delete or update the data in the related source, the data in the related destination can also be deleted or updated automatically. Accordingly, this study adopted the latter design specification to ensure data integrity.

Table 1. Mapping rule about simpletype.

<table>
<thead>
<tr>
<th>Express SimpleType</th>
<th>SqlServer data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>float</td>
</tr>
<tr>
<td>INTEGER</td>
<td>int</td>
</tr>
<tr>
<td>NUMBER</td>
<td>numeric</td>
</tr>
<tr>
<td>LOGICAL (TRUE, FALSE, UNKNOWN)</td>
<td>bit</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>bit</td>
</tr>
<tr>
<td>STRING</td>
<td>nvarchar</td>
</tr>
<tr>
<td>BINANY</td>
<td>varbinary</td>
</tr>
</tbody>
</table>

![Figure 2. Example of IfcCartesianPoint table.](image)

6.3. SelectType

Attributes that are selective types are more special than other data types and their values can be applied to multiple data types highlighted in the definition. Accordingly, it is difficult to perfectly represent attributes of selective types in a RDB, given the inability to specify what specific data type is selected at the DB table design stage. To cope with this problem, this study proposes a method to represent attributes of a selection type by dividing them into two columns, one for the name of the selected data type and the other for the actual value as a character type. Also, if the selectable data type is an entity type, the primary key of the entity instance associated with the column that stores the value is stored. Although this is a simple design technique, the disadvantage is that the DB side cannot guarantee the status of the instances of associated entities because foreign keys cannot be set. Accordingly, maintaining data integrity is contingent on checking against selective IFC data on the application side.

7. About the tool to auto-convert IFC to RDB

To determine the best table design method for converting IFC to RDB, this study developed a
tool to automatically convert from the full IFC4 schema using various table design methods. The tool has three functions: automatic creation of an IFC-based RDB, import of IFC data and QUERY performance testing.

The problem of converting IFC schema to RDB is how to convert the content defined in the EXPRESS language to SQL for creating an RDB. Accordingly, it is necessary to analyze the IFC schema file before the actual conversion. In this study, an analytical model for analyzing IFC schema was devised as shown in the figure 3. First, the BaseType class is defined as the base class for all data types. From this class, we derived classes such as SimpleType, Entity, WrappedType, EnumType, SelectType, AttributeType, etc. The Entity class contains four classes: Attributes, IsAbstract, Subs and Supers. The attributes defined for each entity by the IFC schema are stored in Attributes and IsAbstract can distinguish whether the corresponding Entity is an abstract type or not. All super-entities or sub-entities of an entity are also stored in Subs and Supers. The WrappedType class refers to a new IFC data type defined by the user based on SimpleType, while the SelectType class has a property to determine whether the data type is an aggregate data type, making it easier to create a separate table for aggregate types, as described above. In the AttributeType class, as well as properties like attribute name and attribute data type name, a property called IsOption exists. Using this makes it possible to determine whether the corresponding IFC attribute is a required item or not, so that DB constraints can be set such that null values are not allowed when converting to SQL. Based on the parsing model, this study developed a parser that automatically converts IFC schema files (.exp) to SQL statements using the C# programming language, utilizing Antlr, a parser generator based on parsing. The actual converted SQL is also shown in the figure 5, 6.

A parser was also developed to import IFC instance data into a DB constructed using various table design methods. QUERY performance tests are performed using this imported IFC data.

Figure 3. The UML of Ifc schema parser.

Figure 4. User Interface about IfcDB test tool.
Figure 5. Result of converting Ifc schema to Sql.
In this study, benchmark tests were conducted on various DBs to clarify how performance differed between DBs constructed by various table design methods. First, pre-prepared IFC data was converted to SQL for benchmarking.

---

**Figure 6. Result of converting Ifc data to SQL.**

8. Experiment

8.1. Design

In this study, benchmark tests were conducted on various DBs to clarify how performance differed between DBs constructed by various table design methods. First, pre-prepared IFC data was converted to SQL for benchmarking.

---

**Table 1. Conversion of Ifc to SQL.**

<table>
<thead>
<tr>
<th>IFC</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO T_IFCIndex VALUES(=1234567890, 'description', '01234567890', '2023-01-01', '2023-01-02', '2023-01-03')</td>
<td></td>
</tr>
<tr>
<td>INSERT INTO T_IFCObjectDefinition VALUES(=1234567890, 'description', '01234567890', '2023-01-01', '2023-01-02', '2023-01-03')</td>
<td></td>
</tr>
<tr>
<td>INSERT INTO T_IFCOrganization VALUES(=1234567890, 'description', '01234567890', '2023-01-01', '2023-01-02', '2023-01-03')</td>
<td></td>
</tr>
<tr>
<td>INSERT INTO T_IFCProduct VALUES(=1234567890, 'description', '01234567890', '2023-01-01', '2023-01-02', '2023-01-03')</td>
<td></td>
</tr>
<tr>
<td>INSERT INTO T_IFCResource VALUES(=1234567890, 'description', '01234567890', '2023-01-01', '2023-01-02', '2023-01-03')</td>
<td></td>
</tr>
<tr>
<td>INSERT INTO T_IFCType VALUES(=1234567890, 'description', '01234567890', '2023-01-01', '2023-01-02', '2023-01-03')</td>
<td></td>
</tr>
<tr>
<td>INSERT INTO T_IFCValue VALUES(=1234567890, 'description', '01234567890', '2023-01-01', '2023-01-02', '2023-01-03')</td>
<td></td>
</tr>
<tr>
<td>INSERT INTO T_IFCObject VALUES(=1234567890, 'description', '01234567890', '2023-01-01', '2023-01-02', '2023-01-03')</td>
<td></td>
</tr>
</tbody>
</table>

... (continued)
files of different sizes were imported into the DB to measure data registration time. Next, we devised 10 types of QUERYs suitable for testing IFC data based on the "BUCKY Benchmark" and measured the execution time taken to retrieve the data by executing these QUERYs. The specific QUERY contents and the purpose of the QUERY test are shown in the table 2. In addition, we also measure the processing time to retrieve all rebar start point information from the state where IFC data is stored in the DB. The entity and entity of eight layers, from IfcShapeRepresentation, IfcSweptDiskSolid, IfcCompositeCurve, IfcCompositeCurveSegment, IfcPolyline, IfcCartesianPoint, to IfcShapeRepresentation, IfcSweptDiskSolid, IfcCompositeCurve, IfcCompositeCurveSegment, IfcPolyline and IfcCartesianPoint. Accordingly, this experiment made it easier for us to compare the performance of each DB for very long search routes with a fixed number of stored data.

### Table 2. Query contents.

<table>
<thead>
<tr>
<th>No</th>
<th>SqlServer data type</th>
<th>Test goal</th>
<th>Query in this study</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SINGLE-EXACT</td>
<td>A simple selection operation, independent of other test items</td>
<td>Get columns with PredefinedType = &quot;column&quot;</td>
</tr>
<tr>
<td>2</td>
<td>HIER-EXACT</td>
<td>Querying data in the inheritance relation</td>
<td>Get columns with PredefinedType = &quot;oo&quot;</td>
</tr>
<tr>
<td>3</td>
<td>SINGLE-JOIN</td>
<td>Querying data using the self-join operation</td>
<td>Get multiple columns with the same &quot;PredefinedType&quot; value</td>
</tr>
<tr>
<td>4</td>
<td>HIER-JOIN</td>
<td>Querying performance of inherited classed</td>
<td>Get multiple columns with the same &quot;Tag&quot; value</td>
</tr>
<tr>
<td>5</td>
<td>SET-ELEMENT</td>
<td>Querying collection-type data that satisfy one condition</td>
<td>Get columns with X=&quot;oo&quot; in the relative coordinate system of columns</td>
</tr>
<tr>
<td>6</td>
<td>SET-AND</td>
<td>Collection-type data that satisfy two conditions connected using the AND relation</td>
<td>Get columns with X=&quot;oo&quot; and Y= &quot;oo&quot; in the relative coordinate system of columns</td>
</tr>
<tr>
<td>7</td>
<td>1HOP-NONE</td>
<td>Querying all pairs of data from two tables/classes, which are not in the inheritance relation</td>
<td>Get OwnerHistory information from all columns</td>
</tr>
<tr>
<td>8</td>
<td>1HOP-ONE</td>
<td>Like Query7, but includes a where clause for querying an attribute from the one side in the one-to-many relation</td>
<td>Get the OwnerHistory information of the column for which the GlobalId is specified.</td>
</tr>
<tr>
<td>9</td>
<td>1HOP-MANY</td>
<td>Like Query7, but includes a where clause for querying an attribute from the one side in the one-to-many relation</td>
<td>Get all columns with CHANGEACTION=&quot;NOCHANGE&quot; in the column's OwnerHistory.</td>
</tr>
<tr>
<td>10</td>
<td>2HOP-ONE</td>
<td>Two joins and one where clause condition</td>
<td>Get the OwningApplication information in the OwnerHistory of all columns.</td>
</tr>
</tbody>
</table>

### 8.2. Sampling procedure

This study will test three IFC data sets of varying size and complexity. The first is a simple frame model comprising some columns, beams, floors and walls. The second is an IFC file that integrates the design, equipment and structural sample models included in Revit 2021 and output using the IFC4 Reference View MVD. The third is a concrete frame BIM model of a 20-story high-rise apartment building, including a rebar model. Details of each file are shown in the figure 7.
8.3. Result

The results of various experiments are shown in the figure 8, whereby the processing for data import takes longer with increasing data for all three methods. When the amount of data is small, P3 is fastest, but we cannot confirm a significant difference between P2 and P4. The more the number of instances increases, the more pronounced the difference becomes. In Sample 3, P3 and P2/P4 differ in performance terms by up to 30 minutes. Meanwhile, Q1 and Q3 are queries designed using attributes defined by the entities themselves. With a limited number of instances, P4 has the fastest processing speed. As the number of instances increases, however, P4 processing time increases significantly. However, beyond a certain amount of data, P4 has the slowest processing time, while P2 and P3 can retrieve data within a virtually constant interval, regardless of the data volume and at relatively similar processing speeds.

Q2 is a query that searches for inherited attributes. When the number of instances is small, P4 has the fastest processing speed, while P2 and P3 processing speeds are virtually the same. When the number of instances increases, P2 and P4 become slower, but P3 speed is almost the same. Consequently, P3 performs best when the data volume is large.

Q4 is a query to retrieve multiple instances with the same inherited attributes; the P2 and P4 processing speed declines with increasing data. P3 processing speed remains the same when the data volume is within a certain range, but declines beyond that range. For data in bulk, P2 and P3 significantly outperform P4.

Q5 and Q6 are queries to retrieve aggregate-type data. As the data volume increases, the processing speed declines for all three methods, but far more so for P4 than P2/P3. The Sample 3 results show that P4 takes more than 1000 ms for a single search, while P3 outpaces P2, but not by much.

The Q7-Q10 and rebar query tests use referential relationships between entities to retrieve data. P3 is the fastest and can retrieve data within fixed intervals, regardless of the increase in data volume. P2 and P4 slow down with increasing data volume, while P2 is slower than P3, but not by much. The maximum speed difference is only 70 ms. The disparity in performance between P4 and P2/P3 rises with increasing data. Based on the Sample 3 results, P4 requires processing time of about 8s in Q10.

![Figure 8. The result of experiment.](image-url)
8. Discussion

Based on the data import results, the P3 method seems fastest, because it covers all attributes in a single table and can store data with only a single SQL execution and a single IFC instance as a record. The P2 method is the fastest because it can store data with only one SQL execution and one IFC instance as a record. It is slower than P3 because the attributes of a single IFC instance must be separated and stored in multiple tables. P4, meanwhile, requires import of as much data as the number of attributes, so the performance declines with increasing data and data types.

Based on the Q1 and Q3 results, P4 needs to retrieve specific instances from the instance table. When handling data in bulk, the number of records in the instance table becomes excessive and retrieving the target instance takes time. This is considered attributable to the P4 performance declining with the volume of data. Conversely, in P2 and P3, the amount of data does not affect the processing speed because the table is created for each instance and specific instances need not be identified.

For P3, the number of instances impacted processing speed for P4, since inherited and self-defined attributes are represented in a single table in the full text and the same amount of data is used, regardless of whether the attribute to be searched is inherited or not. In P2, to search for inherited attributes, a JOIN process must be used to retrieve information on the source table from the destination table, which may explain the slightly lower performance than in P3.

In P4, data of the same data type is stored in a single table Q4, resulting in the number of data types of stored IFC data affecting performance. In P2 and P3, meanwhile, data is stored in each entity in a distributed manner. Furthermore, in P2, there are fewer fields for a single table than in P3 because a table for abstract-type entities also exists. Accordingly, in Sample 1, the retrieval speed of P2 exceeds that of P3. However, with increasing data, the JOIN effect becomes more pronounced, which may explain why P2 is slightly slower than P3 in Samples 2 and 3.

Since all three cases use the same method to store aggregate-type data, it is not possible to reflect on how the DB design method affects performance. However, the Q5 and Q6 results suggest that P2 is better in terms of data integrity, since there is no significant difference in performance when comparing P2 and P3.

Based on the Q7-Q10 results and the rebar query test, when retrieving data using the reference relationship between entities, P4 requires a round-trip search to three tables: the table that stores instances in an integrated manner, the table that stores attribute information and the table that stores actual values. This is because the longer the search route, the more significant the performance degradation. In P3, since the related source table stores the related destination table name and ID, search efficiency is considered optimal, since only the related destination and two related source tables can be processed by JOIN in a long search route. However, since foreign keys cannot be set, it is not possible to update data in the source table as data changes in the destination table. Numerous tables are joined. Accordingly, search efficiency is slower than in P3 and the disparity in performance tends to increase as the hierarchy of related entities increases.

Many other factors also affect DB performance. For example, the attributes of the entities to be queried, SQL algorithms, PC specs, etc. Based on the results of this study, the "entity-to-table" method is considered more appropriate for constructing IFC servers than "attribute-to-table." Among the "entity-to-table" methods, P3 outperforms P2 when it comes to importing IFC data and querying via relationships between entities. However, there is no significant difference between the two in terms of retrieving simple entity attributes. P2 is also better able to reliably maintain data consistency and integrity. Since the IFC server services the building lifecycle, we consider data consistency outweighs performance. Accordingly, we believe using P2 as the basic method and combining it with methods that improve query performance is most appropriate method for building an IFC server at present.

9. Concluding remarks

This study focused on database technology, one of the key technologies to realize the construction of a BIM server and developed a suitable storage method for IFC product models in RDB. To date, many researchers have shown that RDBs have low affinity with IFC product models in terms of processing speed and cannot perform well. However, with proper RDB design, IFC data can be normalized to the maximum extent. RDBs are considered more effective than other DBs for data checking when it comes to integrating cumbersome BIM data.
In Japan, IFC models are currently expressed using Part 21 files and data is mainly exchanged on a file basis. As BIM continues to develop, however, there is growing momentum to utilize BIM data while sharing it on a cloud server. We intend to continue our research to realize such an ideal.

Declaration of competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References


抄録

本研究は、BIM データを利活用する可能性を広げるために、Building Information Model がデータサービスの提供者になることを目指すべきと提案する。このような BIM サーバーを利用することで、建築業界内では、煩雑な BIM データを統合・共有できるののみならず、他業界への情報連携することも可能になる。BIM サーバーを構築するために、本研究では、国際標準仕様である IFC をデータベース化するための設計手法考案し、それに伴う問題点を明らかにした。IFC を各種マッピングルールによって RDB を自動作成するプログラムを開発した。さらに、ベンチマークテストを行うことにより、各 DB の性能差や動作を比較して IFC データの格納に適切な RDB 設計手法を明確にした。結果、エンティティ毎にテーブルを作成し、外部キーを持たすことにより、継承関係を保持する方法は、現状 IFC サーバーの構築に最も相応しい手法と考えている。今後 BIM の発展を進める中で、クラウドサーバー上で BIM データを共有しながら活用する機運が高まっている。そのような理想を実現に貢献する。