Optimizing the process of ER diagram creation with PlantUML

Anatolii O. Kurotych, Lesia V. Bulatetska

Lesya Ukrainka Volyn National University, 9 Bankova Str., Lutsk, 43025, Ukraine

Abstract

Relational databases are widely used for storing and processing structured information, while their visualization is a key phase in the architectural design of information systems. One of the most popular tools for creating ERD is PlantUML, which implements the "diagram as code" approach for generating diagrams. This research analyzes PlantUML's capabilities for building ERD and explores tools that improve the diagrams' quality and automate their generation process. The shortcomings of PlantUML's basic functionality for creating ERD are described. Additional PlantUML features are described, such as improving the appearance and readability of diagrams by highlighting primary and foreign keys, removing unnecessary elements, and creating legends for user convenience. A plugin module has been developed to enhance the readability of PlantUML code (hereafter referred to as 'PUML code') by structuring it into functions and procedures, making diagrams easier to create and maintain. The benefits of the modular approach include standardized styles and a minimized amount of PUML code. The article also introduces the Sqlant tool, which allows PUML code to be automatically generated from a PostgreSQL relational database. The PlantUML, combined with automation tools like Sqlant and a modular approach, is an effective tool for producing high-quality ERD. It can be particularly beneficial in environments where database structures undergo frequent changes. Despite the limitations of the official documentation, PlantUML has significant potential to extend its functionality.

Keywords

entity relationship diagram (ERD), computer-aided software engineering (CASE), PlantUML, automatization, relational databases, PostgreSQL, command-line interface (CLI)

1. Introduction

Relational databases have become a standard in software development due to their ability to help solve a wide range of tasks, especially given the popularity of web technologies. As the use of relational databases has grown, so has the number of professionals requiring database skills. Today, SQL knowledge is essential for software engineers, as well as technical support specialists, data analysts, business analysts, marketers, and other IT professionals. Teams of specialists often interact through internal wiki systems such as Confluence, Notion, etc., where information about projects and products, including database schemas, is shared.

ERDs are used to design databases [1, 2]. The process of building ERD is an important stage in software development, as it provides structured data visualization that contributes to the creation of high-quality information systems. Computer-aided software engineering (CASE) tools are often used to create ER diagrams. CASE tools support the automation of the database modeling process and allow the model to be changed flexibly during the development process. Many companies prefer open-source software tools that allow them to adapt to specific needs and integrate with other systems. However, choosing the right CASE tool can be difficult due to the large number of options available. Database modeling tools can be divided into two main categories based on how users interact with them to create diagrams: GUI-oriented or code-as-diagram.

Tools such as Lucidchart [3], ER/Studio [4], SQL Developer Data Modeler [5], draw.io (diagrams.net) [6], MySQL Workbench [7], Microsoft Visio [8], and DBeaver [9] are GUI-oriented. They provide a

CS&SE@SW 2024: 7th Workshop for Young Scientists in Computer Science & Software Engineering, December 27, 2024, Kryvyi Rih, Ukraine

akurotych@gmail.com (A. O. Kurotych); bulatetska.lesya@vnu.edu.ua (L. V. Bulatetska)
https://kurotych.com/ (A. O. Kurotych); https://tinyurl.com/5n6ndd33 (L. V. Bulatetska)
0009-0006-8186-4063 (A. O. Kurotych); 0000-0002-7202-826X (L. V. Bulatetska)

graphical interface for creating diagrams, which better suits less technical users. Tools that use textual descriptions (code-as-diagram approach) allow users to create database models using a specialized code-like language [10]. One of the biggest advantages of this approach is how easily it integrates with version control systems such as Git. It allows storing diagrams directly with the code, greatly facilitating the process of tracking changes, managing versions, and ensuring effective team collaboration [11]. In addition, changes can be made by editing the text directly, without the need to manipulate graphical elements.

PlantUML is one such tool. PlantUML is an open-source tool for generating diagrams based on text descriptions (PUML code) that allows simplifying the process of visualizing the physical model of relational databases [12, 13, 14, 15, 16]. It is available both as a plugin for many development environments (IDEs) [17, 18, 19] and as standalone software [13]. This makes it an ideal choice for engineers who value the flexibility, speed, and simplicity of a text-based approach. In contrast, tools focused on graphical interfaces do not offer the same degree of automation and integration [20].

Despite its wide functionality, PlantUML is insufficiently documented at the official level, which creates difficulties in its use. Therefore, the issues of improving the quality of diagrams and optimizing the process of their generation require additional research [21]. The relevance of this work lies in the need to expand the study of PlantUML capabilities to more effectively display the physical model of databases and explore ways to improve methods for generating ERD, especially in large projects with frequent changes in database structure. In addition, automating the creation of PUML code directly from the database will not only reduce the time required to develop diagrams, but also increase their accuracy and relevance, allowing you to instantly reflect changes. This approach will make PlantUML even more efficient for development teams, providing integration with existing databases and documentation systems.

The research aims to explore the capabilities of PlantUML, develop improved approaches to creating ERDs to provide more convenient and accurate modeling of relational databases and develop an automated system for generating PUML code directly from an existing database.

2. Methods and tools

PUML code was tested on the official PlantUML website (https://www.plantuml.com/plantuml/uml). GitHub (developer platform) was used to host the PlantUML module. SQL queries were checked using RDBMS PostgreSQL version 15.0.

Docker (version 27.3.1) was used to spin up the local PlantUML server (version 1.2024.8) in the section 7.

3. Improving the PlantUML diagram appearance

The study is based on PlantUML diagram types such as the “Entity Relationship Diagram” which, as stated in the documentation, is an extension of the “Class Diagram” with support for relationship descriptions like “Information Engineering Relations”. Figure 1 shows an example of an ERD, taken from the documentation [12].

Let’s take a PostgreSQL database and create the “customer” and “customer_order” tables, which are connected using a foreign key. Additionally, define and use the enumerated type “shipping_method” (figure 2).

3.1. Entity

According to the PlantUML documentation [12, 13, 14], the “customer_order” entity for the ER diagram can be described in the PlantUML language as shown in figure 3.

The PlantUML documentation suggests using the “*” symbol as a visibility modifier to indicate mandatory attributes [12, 13, 14]. In our case, these are all fields that have the “NOT NULL” constraint.

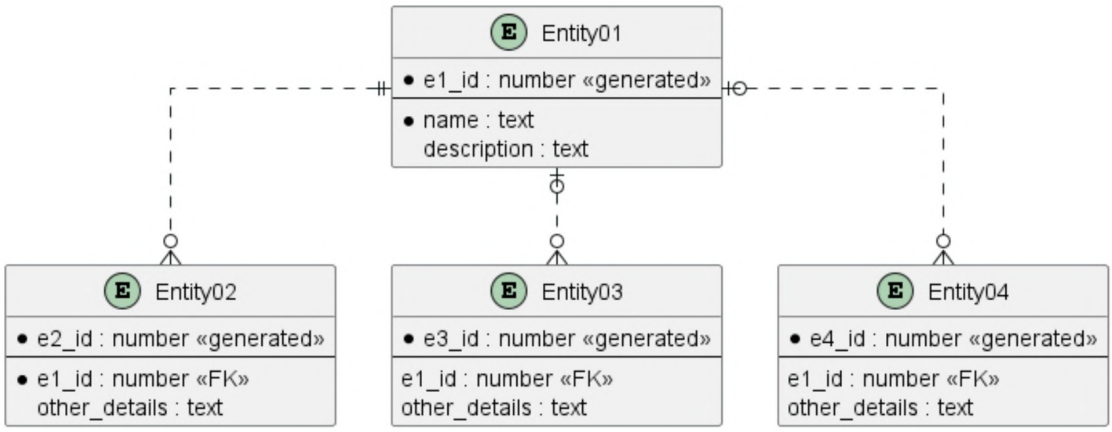


Figure 1: View of the ERD taken from the documentation [12].

```
CREATE TYPE shipping_method AS ENUM ('standard', 'express', 'international');
CREATE TABLE customer (
        id bigserial PRIMARY KEY,
        address text NOT NULL,
        name text NOT NULL,
        registered_at timestamp with time zone  NOT NULL
);
CREATE TABLE customer_order (
  id bigserial PRIMARY KEY,
  customer_id bigint NOT NULL,
  shipping_method shipping_method,
  shipping_address text NOT NULL,
  total_price numeric NOT NULL,
  ordered_at TIMESTAMP WITH TIME ZONE NOT NULL,
  comment_to_order TEXT,
  FOREIGN KEY(customer_id) REFERENCES customer (id)
);
```

Figure 2: SQL script to create the “customer” and “customer_order” tables and the “shipping_method” enumerated data type.

Since “Entity Relationship Diagram” (in PlantUML) is intended not only for displaying schemes of relational databases, such a view (Figure 3) has a number of disadvantages:

- A circle with the letter “E” to the left of the table name has no information value in our case.
- Foreign keys are weakly highlighted, they are usually distinguished by colors or additional symbols.
- “*” symbol, which indicates that a field is mandatory (NOT NULL constraint) and depicted as a black circle to the left of the variable name, may be confusing to a diagram user unfamiliar with PlantUML.
- The same font style for the variable name and its type makes them difficult to distinguish.

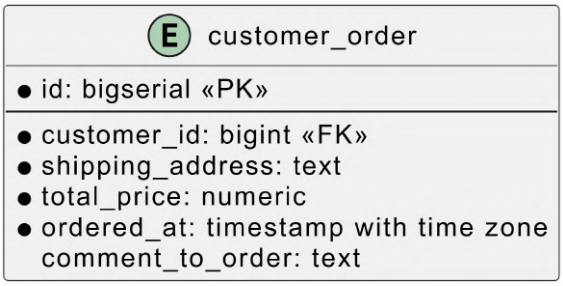
According to the authors’ studies [22, 23], syntax highlighting plays an important role in working with source code. Therefore, a number of the shortcomings described above are considered to be fair.

Using the additional features of PlantUML, the following result can be achieved (figure 4). In figure 4(b), the following rendering flaws have been fixed:

- Deleted letter (E) near the name of the table (using the command “hide circle”).
- The primary key is represented by the “golden key” symbol.

```
@startuml
entity "customer_order" {
  * id: bigserial <<PK>>
  --
  * customer_id: bigint <<FK>>
  * shipping_address: text
  * total_price: numeric
  * ordered_at: timestamp with time zone
  comment_to_order: text
}
@enduml
```

(a)



(b)

Figure 3: PUML code (a) and view of the customer order entity (b) according to PlantUML documentation.

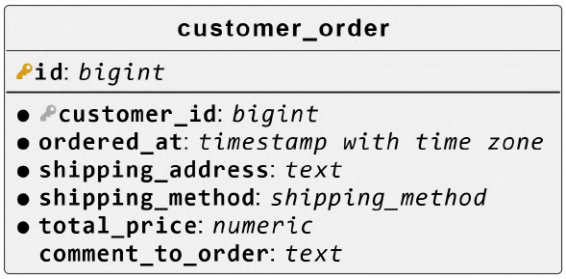
- The foreign key is represented by the “gray key” symbol.

Among the disadvantages, it should be noted that the PUML code in figure 4(a) has become more ‘disorganized’ and less readable.

```
@startuml
hide circle

entity "**customer_order**" {
  <b><color:#d99d1c><&key></color><b>""id""</b>: //""bigint"" //
  ---
  *<color:#aaaaaa><&key></color><b>""customer_id""</b>: //""bigint"" //
  *<b>""ordered_at""</b>: //""timestamp with time zone"" //
  *<b>""shipping_address""</b>: //""text"" //
  *<b>""shipping_method""</b>: //""shipping_method"" //
  *<b>""total_price""</b>: //""numeric"" //
  <b>""comment_to_order""</b>: //""text"" //
}
@enduml
```

(a)



(b)

Figure 4: PUML code (a) after improvement and the rendered result (b).

3.2. Enum

Most SQL databases (MySQL, PostgreSQL, Oracle, etc.), as well as most programming languages, support the enum data type. Using the “entity” keyword creates an unwanted artifact during rendering the diagram (figure 5, marked with a red arrow).

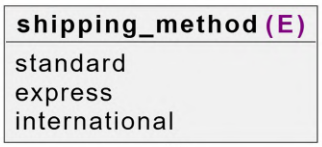
```
@startuml
hide circle
entity shipping_method {
  standard
  express
  international
}
@enduml
```

(a) (b)

Figure 5: PUML code (a) and an unwanted artifact when using the “entity” keyword to create an enum data type in PlantUML (b).

To remove this artifact, the ‘object’ keyword should be used. Additionally, highlighting the name of the list by adding the identifier ‘(E)’ can help make it easier to distinguish from other objects (figure 6).

```
@startuml
object "**shipping_method** <color:purple>**(E)**</color>" as product_category {
 standard
 express
 international
}
@enduml
```



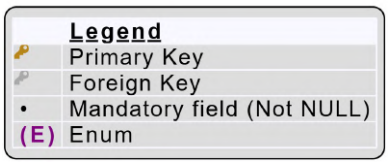
(a) (b)

Figure 6: Enum PUML code (a) and rendered result (b).

3.3. Legend

Since some symbols (as mentioned above in disadvantages) may not be clear to the end user, a “legend” can be used to explain their meaning. As the diagram becomes more complex, the legend can be expanded. For example, in figure 7 shows four objects in the legend: Foreign Key, Primary Key, Not NULL, and Enum.

```
legend right
 <#GhostWhite,#GhostWhite>|   |= __Legend__ |
 |<b><color:#b8861b><&key></color></b>| Primary Key |
 |<color:#aaaaaa><&key></color>| Foreign Key |
 | &#8226; | Mandatory field (Not NULL) |
 | <color:purple>**(E)**</color> | Enum |
endlegend
```



(a) (b)

Figure 7: PUML code (a) and appearance of the legend (b).

3.4. Overview of types of relationships

PlantUML uses the relationship description based on the Information Engineering notation. Figure 8 shows the types of relationships that are supported.

Also, it is worth mentioning that it is possible to add a comment text to each relationship (figure 9).

Type	Symbol
Zero or One	`|o--`
Exactly One	`||--`
Zero or Many	`}o--`
One or Many	`}|--`

Figure 8: Types of relationships supported by PlantUML.

```
customer_order }o--|| customer : "No more than 100 items per order per customer"
```

Figure 9: Comment to relationship.

4. Improving the quality of PlantUML code

The readability of the PUML code is an important characteristic because it is responsible for rendering the diagrams and can be read and edited by different team members. To improve the readability and convenience of writing PUML code, it's worth reducing its volume. This can be achieved by using built-in functions (!function) and procedures (!procedure) or by creating your own, which are organized into plugin modules. It is important to note that there are built-in functions beginning with the symbol '%'. You can find their brief descriptions on the official PlantUML website in the 'Builtin functions' section [13]. One such function, '%splitstr', which splits the text into an array of strings, was used in the following example for the 'enum' procedure. In PlantUML, functions differ from procedures in that functions must return a result. Additionally, as in other programming languages, both functions and procedures can accept parameters. Figure 10 shows two functions (column, table) and their usage. The result of this code is shown in figure 4.

Figure 11 shows the enum procedure and its usage. The result of this code is shown in figure 6(b).

As can be seen from figures 10 and 11, the amount of code has increased rather than decreased because the example only shows one entity that uses custom functions. However, when there are more tables, this approach will be more efficient.

5. Designing a plugin module for PlantUML

Procedures and functions that are used for the design of ERDs should be placed in a separate file with the .puml extension and included in the main file using the "!include" command. This approach provides modularity and improves code organization. The module file db_ent.puml was created with the next content:

- Functions: table($name), column($name, $type, $pk=false, $fk=false, $nn=false);
- Procedures: enum($name, $variants), add_legend().

To use the module, it must be placed on any web resource so that it has a permanent HTTP(s) address and is available to users. In this study, a public GitHub repository was used to host the PlantUML module. It can be accessed at the following address: https://raw.githubusercontent.com/kurotych/sqlant/b2e5db9ed8659f281208a687a344b34ff38129cd/puml-lib/db_ent.puml After the PlantUML module is hosted on web resource there is a possibility to include it in PUML code (figure 12).

This approach gives us the following advantages:

- PUML code file contains only the logic of the diagram without unnecessary code-specific details, which improves the code readability.
- Generic code can be reused in other files without the need for repetition.
- Changes in the PlantUML module are automatically reflected in all related diagrams, which is useful for fixing bugs.

```
@startuml
hide circle

!function column($name, $type, $pk=false, $fk=false, $nn=false)
  !local $prefix = ""

  !if ($pk == true)
     !$prefix = "<color:#d99d1c><&key></color>"
  !elseif($nn == true)
     !$prefix = "*"
  !endif

  !if ($fk == true)
     !$prefix = $prefix + "<color:#aaaaaa><&key></color>"
  !endif

!return $prefix + '<b>""' + $name + '""</b>' + ': ' + '//""' + $type + '"" //'
!endfunction

!function table($name)
  !return 'entity "**' + $name + '**"'
!endfunction

table(customer_order) {
  column(id, "bigint", $pk=true)
  ---
  column(customer_id, "bigint", $fk=true, $nn=true)
  column(ordered_at, "timestamp with time zone", $nn=true)
  column(shipping_address, "text", $nn=true)
  column(shipping_method, "shipping_method", $nn=true)
  column(total_price, "numeric", $nn=true)
  column(comment_to_order, "text")
}
@enduml
```

Figure 10: Definitions of “column” and “table” functions and their usage.

```
@startuml
!procedure enum($name, $variants)
  !$list = %splitstr($variants, ",")

  object "**$name** <color:purple>**(E)**</color>" as $name {
     !foreach $item in $list
     $item
     !endfor
  }
!endprocedure

enum("shipping_method", "standard, express, international")
@enduml
```

Figure 11: Enum procedure and its usage.

- The standardization of styles and procedures for all project participants simplifies teamwork.

This approach makes it easy to scale projects and keep code clear and organized. Designing functions (and procedures) in a module will provide an advantage even in the case with a single entity in a diagram.

6. Automation of PlantUML code generation for PostgreSQL RDBMS

A software tool called Sqlant (https://github.com/kurotych/sqlant, [24]) was developed to automate the generation of PlantUML code. Starting from version 0.4, it implements the diagram generation

```
@startuml
hide circle
!include
https://raw.githubusercontent.com/kurotych/sqlant/b2e5db9ed8659f281208
a687a344b34ff38129cd/puml-lib/db_ent.puml
```

Figure 12: Including an external PUML module into code.

approach described in this study. It can be installed using Cargo (a package manager for Rust projects) [25]. For users of the GNU/Linux operating system, a statically compiled binary can be downloaded at https://github.com/kurotych/sqlant/releases [24]. Sqlant is a CLI (command-line interface) tool that generates PUML code (mermaid generation is also supported [26]) based on a connection string. The result is output to stdout, which can be redirected to a file. Figure 13 shows the sequence diagram of the engineer's workflow with the Sqlant tool. Sqlant requires one mandatory parameter: the connection string (CON_STRING), which contains the database address and authentication data. The database must be accessible for connection from the Sqlant runtime.

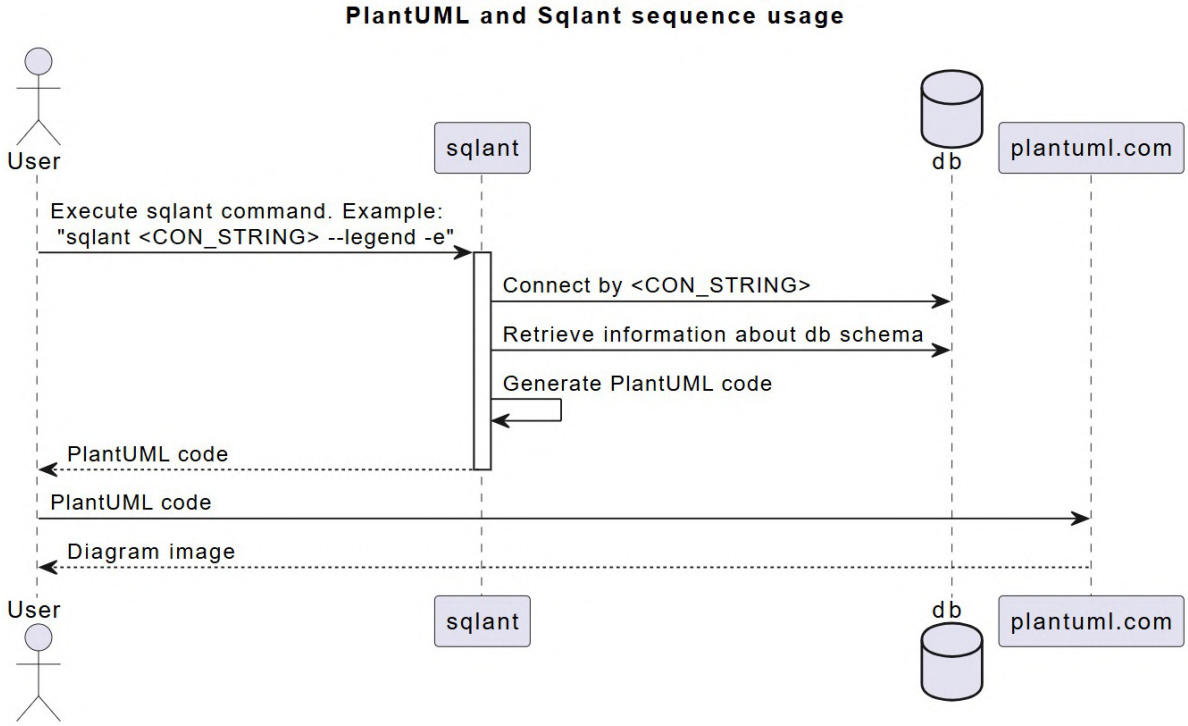


Figure 13: PlantUML and Sqlant sequence usage.

The internal Sqlant workflow operates as follows:

1. Retrieve information about the database schema, including table names, columns, foreign keys, and other related metadata.
2. Generate PUML code based on the retrieved information.
3. Output the PUML code to standard output, which can be redirected to a file using standard OS utilities.

After that, the user executes the received code on the PlantUML server side and receives the result in the form of a diagram. The generated diagram can be downloaded in several formats: PNG, SVG, ASCII Art. This tool supports several options for customizing the result. After installation, users can familiarize themselves with these options by using the `--help` argument (figure 14).

```
❯ sqlant --help
Generate Entity Relationship diagram textual description from SQL connection string

Usage: sqlant [OPTIONS] <connection_string>

Arguments:
  <connection_string>

Options:
      --inline-puml-lib  Inline PlantUML lib into diagram code
      --legend           Add legend to diagram (supported only for PlantUML)
  -n, --nn               Add NOT_NULL(NN) marks (for PlantUML always true)
  -e, --en               Draw enum types
  -s, --schema <schema>  Schema name [default: public]
  -o, --output <output>  Generate output in mermaid format [default: plantuml] [possible
values: plantuml, mermaid]
  -h, --help             Print help
  -V, --version          Print version
```

Figure 14: Output of the - -help command in Sqlant.

7. Data security: installing the PlantUML server locally

It is worth noting that in the above examples, the plantuml.com site was used to convert PUML code into PNG/SVG format. This site may (potentially) save these diagrams, as PlantUML generation occurs on the server side. This could be unacceptable for many companies, as it may expose the internal architecture of the database to third parties. Since PlantUML is an open-source product, its source code can be reviewed, and it can be installed locally. One option for installation is using the Docker containerization system. The following command will launch the PlantUML server (https://github.com/plantuml/plantuml-server) with a web interface accessible locally at the address: http://localhost:8080 (figure 15).

docker run -d -p 8080:8080 plantuml/plantuml-server:jetty

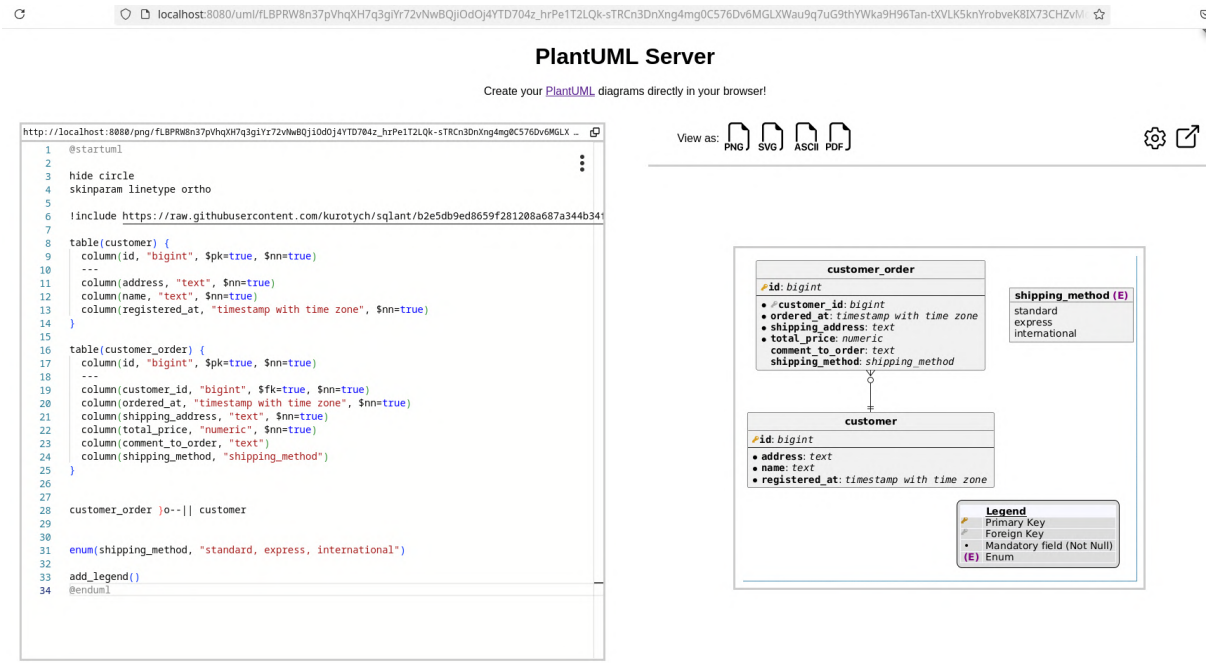


Figure 15: Local PlantUML server.

When replacing the use of plantuml.com with a locally installed plantuml.jar file (https://plantuml.com/download) or a locally installed Docker container running the PlantUML server, this approach becomes easily automatable. As a result, some engineers integrate this process into their bash scripts, which can then be used in the CI (Continuous Integration) workflow (figure 16) [27].

```
16    cargo install sqlant && sudo apt install -y openjdk-11-jdk && sudo apt install -y graphviz
17
18    # use sqlant and plantuml to extract erd from any db
19    sqlant postgresql://postgres:$PASSWORD@localhost/hoopoe > $(pwd)/infra/hoopoe.uml
20    java -jar $(pwd)/infra/plantuml.jar $(pwd)/infra/hoopoe.uml
21
```

Figure 16: An example of automation (part of a bash script) [27].

8. Conclusions

During the study of PlantUML capabilities, the main functions of this tool for creating ERDs were analyzed. An approach is proposed, which includes the development of a plugin module to simplify the process of modeling the physical structure of databases. The module allows you to standardize code, improve readability, reduce duplication, and facilitate teamwork. In addition, automation of the PUML code generation process from the PostgreSQL relational database was implemented. The Sqlant tool was developed for automation purposes [24]. Sqlant retrieves information about the database schema, generates PUML code that can be used to visualize the ERDs in the PlantUML environment. Despite the lack of official documentation, PlantUML has shown significant potential for integration into the work of development teams due to its flexibility, modularity, and open-source nature.

Declaration on Generative AI: During the preparation of this work, the author used ChatGPT in order to: Grammar and spelling check. After using this tool/service, the authors reviewed and edited the content as necessary and take full responsibility for the publication's content.

References

[1] D. Martínez, L. Po, J. R. R. Trillo-Lado, R.and Viqueira, A conceptual data modeling framework with four levels of abstraction for environmental information, Environmental Modelling and Software 183 (2025) 1–23. doi:10.1016/j.envsoft.2024.106248.

[2] S. Hettiarachchi, C. Sugandhika, A. Kathriarachchi, S. Ahangama, G. T. Weerasuriya, A Scenario-based ER Diagram and Query Generation Engine, in: B. H. Sudantha, R. D. Wageeshani (Eds.), Proceedings of the 4th International Conference on Information Technology Research (ICITR), Information Technology Research Unit Faculty of Information Technology University of Moratuwa Sri Lanka, IEEE, Moratuwa, Sri Lanka, 2019, pp. 1–5. doi:10.1109/ICITR49409.2019.9407793.

[3] Lucid Software Inc., Documentation and Implementation Level 1 | Lucidchart, 2025. URL: https://www.lucidchart.com/pages/templates/documentation-and-implementation-level-1.

[4] ER/Studio Data Architect Product Documentation - Embarcadero Technologies, 2024. URL: https://docs.embarcadero.com/products/er_studio/.

[5] Oracle, SQL Developer Data Modeler Documentation Release 18.1, 2025. URL: https://docs.oracle.com/database/sql-developer-data-modeler-18.1/.

[6] JGraph Ltd, draw.io Documentation, 2023. URL: https://www.drawio.com/doc/.

[7] Oracle, MySQL :: MySQL Workbench Manual, 2025. URL: https://dev.mysql.com/doc/workbench/en/.

[8] Microsoft, Visio help & learning, 2025. URL: https://support.microsoft.com/en-us/visio.

[9] DBeaver Documentation, 2024. URL: https://dbeaver.com/docs/dbeaver/.
[10] Terrastruct, Text to diagram, 2025. URL: https://text-to-diagram.com/.
[11] The Mibex Software team, Optimizing documentation with PlantUML - Integration, Efficiency, and Best Practices, 2024. URL: https://mibexsoftware.com/blog/plantuml/.
[12] Drawing UML with PlantUML: PlantUML Language Reference Guide (Version 1.2023.11), 2023. URL: https://pdf.plantuml.net/PlantUML_Language_Reference_Guide_en.pdf.
[13] Open-source tool that uses simple textual descriptions to draw beautiful UML diagrams, 2025. URL: https://plantuml.com/.
[14] The Hitchhiker's Guide to PlantUML documentation, 2020. URL: https://crashedmind.github.io/PlantUMLHitchhikersGuide/.
[15] E. Gosselin, PlantUML for database modeling, 2024. URL: https://medium.com/@elvis.gosselin/plantuml-for-database-modeling-1b71e6d4622d.
[16] R. Léger, SQL + PlantUML: Generate Automatic Database Diagrams, 2020. URL: https://tinyurl.com/2bm52k8j.
[17] A. Akundi, J. Ontiveros, S. Luna, Text-to-Model Transformation: Natural Language-Based Model Generation Framework, Systems 12 (2024) 369. doi:10.3390/systems12090369.
[18] J. Lund, L. B. Jensen, N. Battle, P. G. Larsen, H. D. Macedo, Bidirectional uml visualisation of vdm models, 2023. doi:10.48550/arXiv.2304.06618. arXiv:2304.06618.
[19] V. Krása, PlantUML Integration - IntelliJ IDEs Plugin | Marketplace, 2024. URL: https://plugins.jetbrains.com/plugin/7017-plantuml-integration.
[20] Appmus, Best PlantUML Alternatives & Reviews, 2021. URL: https://appmus.com/software/plantuml.
[21] D. Rouabhia, I. Hadjadj, Enhancing Class Diagram Dynamics: A Natural Language Approach with ChatGPT, 2024. doi:10.48550/arXiv.2406.11002. arXiv:2406.11002.
[22] T. R. Beelders, J.-P. L. du Plessis, Syntax highlighting as an influencing factor when reading and comprehending source code, Journal of Eye Movement Research 9 (2016). doi:10.16910/jemr.9.1.1.
[23] A. Sarkar, The impact of syntax colouring on program comprehension, in: Proceedings of the 26th Annual Conference of the Psychology of Programming Interest Group, PPIG2015, Bournemouth, 2015, pp. 49–58. URL: https://ppig.org/files/2015-PPIG-26th-Sarkar1.pdf.
[24] A. Kurotych, GitHub - kurotych/sqlant: Generate PlantUML/Mermaid ER diagram textual description from SQL connection string, 2024. URL: https://github.com/kurotych/sqlant.
[25] The Cargo Book, 2025. URL: https://doc.rust-lang.org/cargo/.
[26] Mermaid | Diagramming and charting tool, 2025. URL: https://mermaid.js.org/.
[27] hoopoe/scripts/setup.sh at 3ac947c657ec989bc36eea897f1a5061518a8429 · wildonion/hoopoe, 2024. URL: https://github.com/wildonion/hoopoe/blob/3ac947c657ec989bc36eea897f1a5061518a8429/scripts/setup.sh#L16.