Computational Technology and Software Tool for Translation of Business Rules into Database Creation Scripts

Andrii Kopp and Dmytro Orlovskyi

National Technical University "Kharkiv Polytechnic Institute", Kyrpychova str. 2, Kharkiv, 61002, Ukraine

Abstract

Almost all modern software systems from small mobile applications to large enterprise suites are using databases and database management systems to store and manage data collections and ensure their persistence and security. However, dominating Agile software development methodologies do not assume engineers focus only on database design or any other specific field. In Agile projects, database design usually belongs to the scope of software developers responsible for server-side programming. Hence, a lack of specialized skills and experience may lead to database design shortcomings or even errors causing time and money expenses at the later stages of software development projects. When software requirements are elicited, database entities and relations are captured in business rules – brief and precise descriptions of a considered domain. Therefore, in this paper, we propose a computational technology and a software tool for the translation of business rules into database creation scripts to reduce time and cost expenses at the development or maintenance stages of software engineering projects. In this study we use the finite-state machine based approach to parse business rules, association rules learning, and naming conventions to detect data types using attribute names, and the most widely-used database management systems to perform experiments.

Keywords

Business Rules, Database Design, Relational Model, Database Schema, Code Generation

1. Introduction

Today is hard to imagine software applications that do not use a database (DB) to keep user data. In general, a database is the collection of raw facts (i.e. end-user data), which is stored in a structured manner, and metadata that describes this structure and how end-user data is managed. Databases are handled by the specialized system software called the Database Management System (DBMS). The DBMS is responsible for controlling the database structure's persistence and security [1].

Currently, software applications of different complexity and data storage requirements, from food delivery mobile applications to large enterprise information systems (IS), keep their data collections in databases managed by DBMS. As a result, database design is a crucial component of practically any software engineering project and may call for specialized engineers with strong expertise in data modeling and DB development. This approach may work for waterfall or iterative projects when task delegation to various specialists or even teams and big project teams are common things [2].

However, according to Agile practices dominating in software engineering projects, DB design is done by the same software engineers responsible for the back-end (i.e. server-side) development, or even by full-stack developers responsible for both front-end and back-end programming [2].

This makes engineers responsible for DB design and development replaceable, while the whole software engineering process is more flexible. Nevertheless, the lack of database design skills, proper training, and experience, as well as the time and staff resources to focus more precisely on database schema correspondence to a domain, may lead to design mistakes and further time and monetary

COLINS-2023: 7th International Conference on Computational Linguistics and Intelligent Systems, April 20–21, 2023, Kharkiv, Ukraine
EMAIL: kopp93@gmail.com (A. Kopp); orlovskyi.dm@gmail.com (D. Orlovskyi)
ORCID: 0000-0002-3189-5623 (A. Kopp); 0000-0002-8261-2988 (D. Orlovskyi)

expenses for fixing errors made at the DB design stage. It is well-known though, that the cost of error fixing increases exponentially during the project and may be several times higher at the development stage than at the design stage [3].

Business rules are used as sources of DB design requirements, helping to detect necessary entities, attributes, and relationships. According to [1], business rules (BR) are textual explanations of policies, procedures, and concepts within a certain organization that are compact, exact, and unambiguous. For example, when the IS or its certain components are planned for development, or when the out-of-the-box enterprise software is planned for customization, BR could be taken from company employees or documents [1].

Database BR are definitions of data model components, including entities, attributes, relationships, and constraints [1]. A similar definition of DB BR is given in Wiegers and Beatty's book devoted to software requirements elicitation [4]. According to their taxonomy, there are six types of BR, such as facts, constraints, action enablers, inferences, computations, and terms. Whereas the other types of BR describe the software system's behavior in action, facts describe data model entities and relationships. Thus, in the rest of the paper, we will consider BR as facts that describe the static structure of data models according to Wiegers and Beatty [4].

Therefore, this paper aims at reducing time losses and related costs occurring at the late stages of software engineering projects when the DB design is handled by inexperienced or untrained engineers responsible for server-side or full-stack development. The goal could be achieved by developing the computational technology and software tool for the translation of BR into DB creation scripts, such as Data Definition Language (DDL) statements of Structured Query Language (SQL).

The rest of this paper is structured as follows: the state-of-the-art analysis is given in sub-section 1.1, the formal problem statement is given in 1.2, materials and methods for DB schema generation from BR are outlined in section 2 and its respective sub-sections, while the software tool development and usage to solve the considered problem is described in section 3. The analysis of obtained results includes validation of SQL scripts generated from BR for compliance with the most popular DBMS. In the last section conclusion and future work in this field are formulated.

2. Related Works

According to the "DB-Engines Ranking" website that ranks DBMSs according to their popularity, the most widely used are relational database management systems: Oracle, MySQL, Microsoft SQL Server, and PostgreSQL [5]. Even though these DBMSs also support other data models, they still use relational models as their primary mechanisms. It is well-known, that relational databases consist of tables, table columns, and relationships that link tables together [6]. Database tables represent entities of a certain domain, table columns represent entity attributes, primary keys uniquely identify records in tables, and foreign keys establish references between tables [7]. As can be seen, the main principles of relational database design did not change much since they were introduced in the 1970s.

Existing studies in the field of automatic DB schema generation from business rules demonstrate different approaches. In [8] authors proposed the automatic generation of an extended entity-relationship (ER) diagram using natural language processing (NLP). This approach by Šuman et. al [8] parses sentences and tags them as various parts of the speech by then mapping these words to ER elements: entities, attributes, and relationships [8]. Another study [9] proposes an algorithm for the ER diagram transformation into relational database schema, however, it uses ER models partitioning into "ER-construct-units" to validate transformations but not to produce SQL code [9]. As for NLP-based approaches, authors of [10] propose a system that detects and extracts necessary information from a given natural language text scenario and builds the ER diagram and SQL queries. However, the system proposed in [10] is focused on generating SQL queries for the data access (i.e. SELECT statements), not database creation. Similarly, authors of [11] proposed a novel natural language interface to databases named SQL Query Generation Engine, which is used to translate natural language into SQL queries to read data from databases [11]. One more interesting paper [12], which is focused on database creation, proposes a new method that uses NLP techniques for UML (Unified Modeling Language) class diagram generation from requirement specifications to allow software developers to analyze requirements in an efficient way [12].

3. Proposed Approach

3.1. Business Rules Processing and Formalization

Let us also represent fact business rules using controlled language structures of limited grammar and complexity. Hence, the structure of fact business rules [1, 4] can be formally described using the Extended Backus-Naur Form (EBNF) [13] as following:

$$< fact >::= \{each\} < entity\_name > \{has\} < attribute\_name > . \quad (1)$$

Hence, a deterministic finite-state machine (DFSM) [14] is proposed to parse a sequence of tokens $Tokens$ included into each business rule:

$$(Tokens, S, s_0, \delta, F), \quad (2)$$

where:

- $Tokens$ is the finite non-empty set of tokens of each business rule;
- S is the finite non-empty set of states;
- s_0 is the initial state, $s_0 \in S$;
- $\delta : S \times Tokens \to S$ is the state-transition function;
- F is the set of final states, $F \subset S$.

The proposed DFSM is given in Fig. 1 below using the UML state diagram.

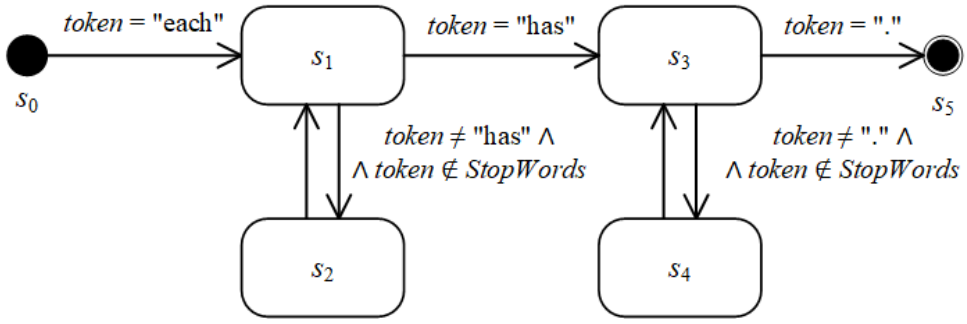


Figure 1: The state diagram of proposed DFSM for business rules parsing

Besides the initial state $s_0 \in S$, the set of states S includes the following ones:

- s_1 is the state to which the DFSM transfers after the “each” token is passed as input;
- s_2 is the state to which the DFSM transfers after the token, which corresponds the following conditions, is passed as input:

$$token \neq \text{"has"} \wedge token \notin StopWords, \quad (3)$$

where $StopWords$ is the set of stop words, which can be filtered using the NLTK (Natural Language Toolkit) package [15];

- s_3 is the state to which the DFSM transfers after the “has” token is passed as input;
- s_4 is the state to which the DFSM transfers after the token, which corresponds the following conditions, is passed as input:

$$token \neq \text{"."} \wedge token \notin StopWords; \quad (4)$$

- s_5 is the final state to which the DFSM transfers after the “.” is passed as input, $s_5 \in F$.

Let us describe the additional behavior of the proposed DFSM:

- when the state machine transfers to s_2, the given token is added to a set that forms the name of entity $EntityName$;
- when the state machine transfers to s_4, the given token is added to a set that forms the name of attribute $AttributeName$.

As given above, the proposed DFSM (Fig. 1) takes fact business rules given according to (1) and produces ER model components: entity names and attribute names. The further problem includes the detection of relationships between extracted entities, as well as data types assignment to attributes.

Let us formally describe a relational database structure using the following equations:

$$\begin{gathered} Entities = \{Entity_i, i = 1,2,\ldots,n\}, \\ \forall i = 1,2,\ldots,n : Entity_i = \left\langle EntityName_i, \{Attribute_k^i, k = 1,2,\ldots,q\} \right\rangle, \\ \forall i = 1,2,\ldots,q : Attribute_k^i = \left\langle AttributeName_k^i, AttributeType_k^i \right\rangle, \\ \delta : Attributes \to DataTypes = \{DataType_j, j = 1,2,\ldots,m\}, \\ AttributeType_k^i \in DataTypes, \\ \rho : Attributes^{\cap} \to \{(Entity^{PK}, Entity^{FK}), Entity^{PK} \in Entities \wedge Entity^{FK} \in Entities\}, \\ Attributes^{\cap} \subseteq Attributes, \end{gathered} \tag{5}$$

where:

- $Entities$ is the set of entities;
- $Entity_i$ is the i-th entity of a database structure, $i = 1,2,\ldots,n$;
- n is the number of entities in a database structure;
- $Attributes$ is the set of attributes;
- $EntityName_i$ is the name of i-th entity, $i = 1,2,\ldots,n$;
- $Attribute_k^i$ is the k-th attribute of i-th entity, $k = 1,2,\ldots,q$;
- $AttributeName_k^i$ is the name of k-th attribute of i-th entity, $k = 1,2,\ldots,q$;
- $AttributeType_k^i$, is the data type of k-th attribute of i-th entity, $k = 1,2,\ldots,q$;
- q is the number of attributes in a database structure;
- $DataTypes$ is the set of all data types that can be used in a database structure;
- $DataType_j$ is the j-th data type, $j = 1,2,\ldots,m$;
- m is the number of data types used in a database structure;
- θ is the function that defines a data type $DataType_j$, $j = 1,2,\ldots,m$ for each attribute;
- $Attributes^{\cap}$ is the sub-set of attributes that form relationships, $Attributes^{\cap} \subseteq Attributes$;
- $(Entity^{PK}, Entity^{FK})$ is the pair of entities that form a relationship;
- $Entity^{PK}$ is the parent entity;
- $Entity^{FK}$ is the child entity;
- ρ is the function that defines a pair of parent and child entities for each common attribute given in the sub-set $Attributes^{\cap}$.

Using the proposed formal description of a relational database structure (5), the input data in the form of business rules (1) processed by the DFSM (Fig. 1) will be structured to generate SQL scripts.

3.2. Preliminary Extraction of Entities and Attributes Using DFSM

Let us consider the example of the following business rules describing a domain of students and courses:

"Each student has a student ID. Each student has a full name. Each course has a course number. Each course has a title. Each course has a number of credits. Each enrollment has a student ID. Each enrollment has a course number. Each enrollment has a grade."

Using the proposed DFSM, we obtain the following entities and their attributes. All input tokens combined to obtain entity names and attribute names values should be merged using the underscore (i.e., "_") separator to simplify the further generation of syntactically correct SQL code. Stop words should be removed according to equations (3) and (4).

The initial information on the database structure, which is extracted from the business rules given above, is demonstrated in Table 1.

Table 1
Extracted entities and attributes from business rules

Entity name	Attribute name
student	student_id, full_name
course	course_number, title, number_credits
enrollment	student_id, course_number, grade

3.3. Relationships Detection

Relationships between entities should be created using common attributes – primary keys (PK) in parent entities and foreign keys (FK) in child entities. To detect relationships, we propose to check all pairs of entities and find intersections of their attributes. An entity that firstly mentions the common attribute should be considered as parent, while other entities – as child (Fig. 2).

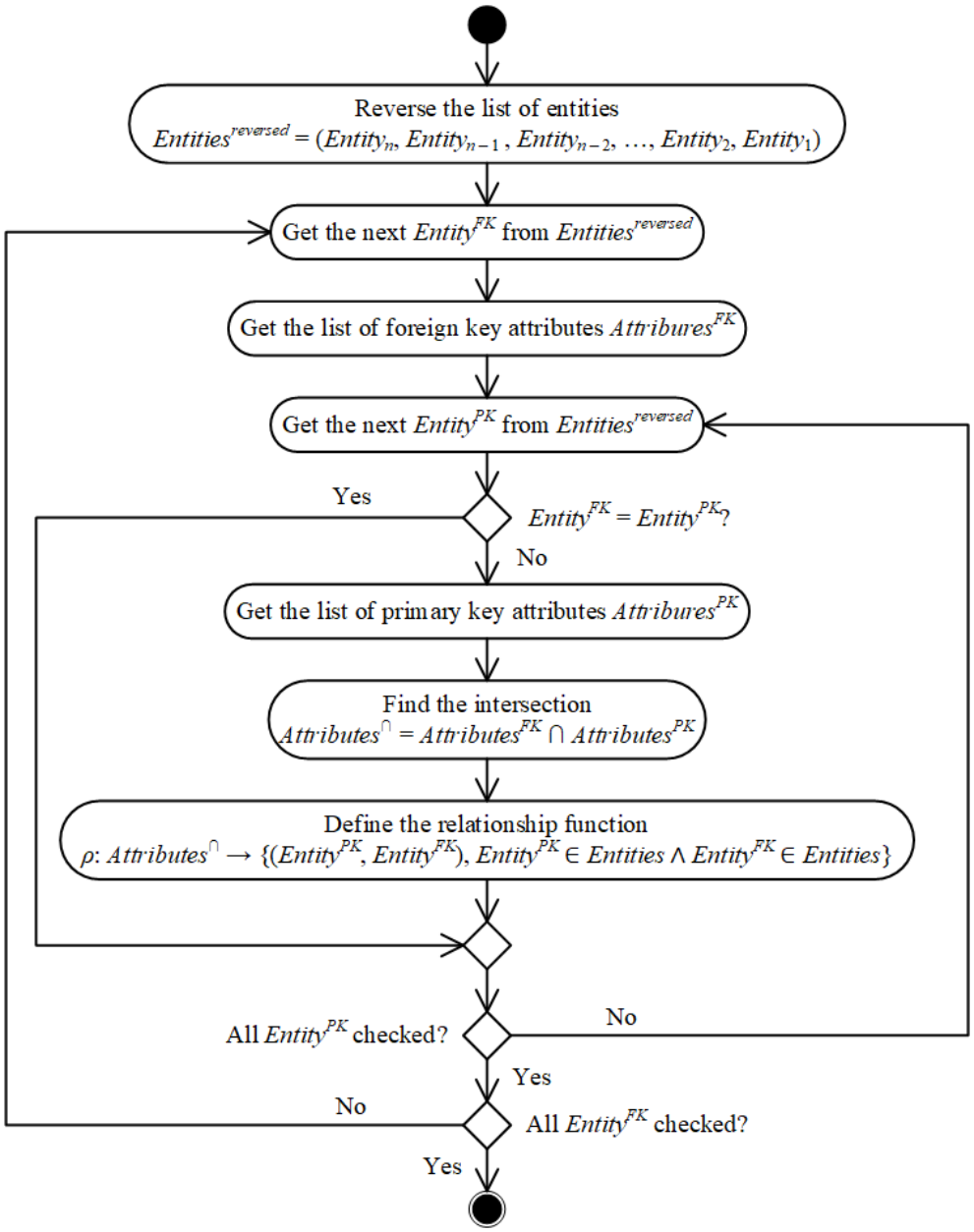


Figure 2: The algorithm for detecting relationships between entities

Using extracted entities and their respective attributes (Table 1), we can apply the algorithm given above (Fig. 2) to detect relationships between entities (Table 2).

Table 2
Detected relationships, parent, and child entities

Common attribute	Parent entity	Child entity
course_number	course	enrollment
student_id	student	enrollment

Found common attributes should be used as PK in parent entities and as FK in child entities when generating SQL code.

3.4. Attribute Data Types Suggestion using Apriori Algorithm

The following essential problem includes the data types suggestion for attributes using their names only. In this study we use the "Spider 1.0 NLP Dataset" created and supported by Yale students. This dataset includes 166 databases with various tables that cover 138 different domains [16].

Using the metadata descriptions of database tables from the "Spider" dataset, we prepared the set of attribute names and corresponding data types. These pairs are used by the association rule learning Apriori algorithm (Fig. 3) [17].

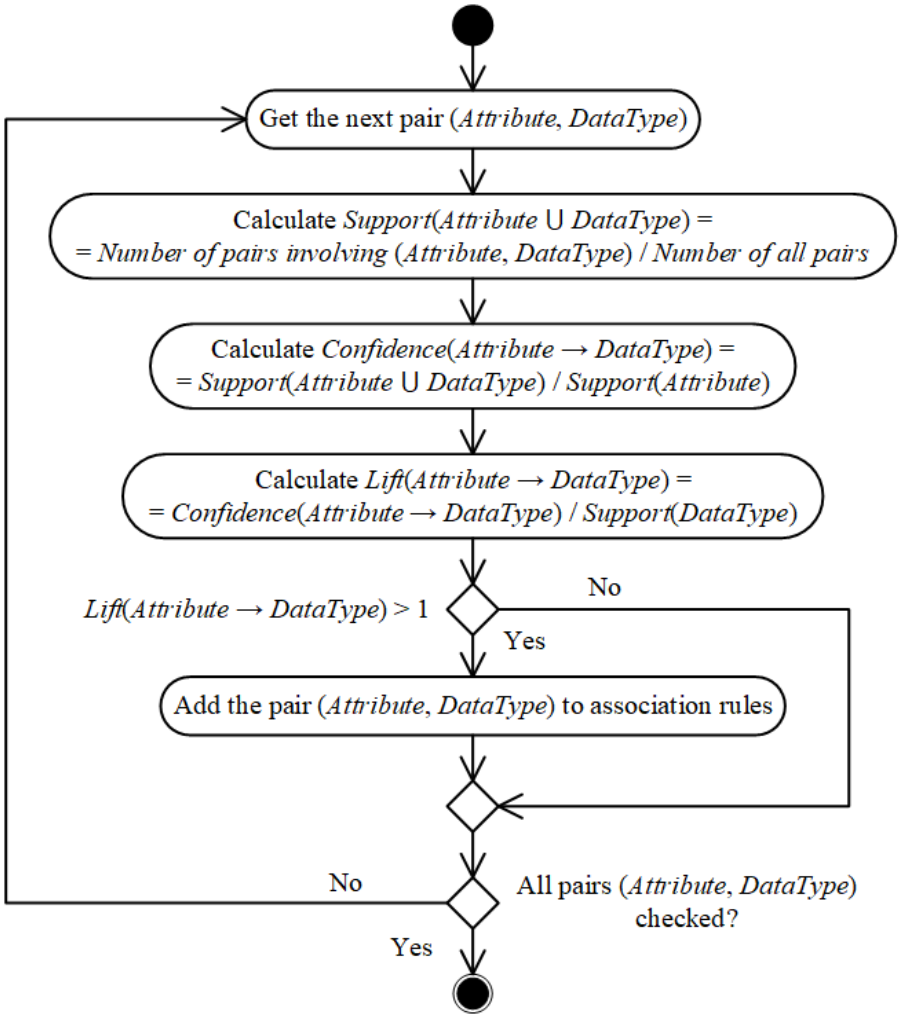


Figure 3: The Apriori association rule algorithm for processing of attribute name and data type pairs

Using the Apriori algorithm [17], the following metrics should be calculated:

$$Support(Attribute \cup DataType) = \frac{\#Pairs(Attribute, DataType)}{\#Pairs}, \quad (6)$$

$$Confidence(Attribute \rightarrow DataType) = \frac{Support(Attribute \cup DataType)}{Support(Attribute)}, \quad (7)$$

$$Lift(Attribute \rightarrow DataType) = \frac{Confidence(Attribute \rightarrow DataType)}{Support(DataType)}, \quad (8)$$

where:

- $Support(Attribute \cup DataType)$ is the probability of observing the attribute and the data type together, $\#Pairs(Attribute, DataType)$ is the number of pairs containing both the attribute and the data type, and $\#Pairs$ is the number of all pairs;
- $Confidence(Attribute \rightarrow DataType)$ is the probability of having the data type for attribute;
- $Lift(Attribute \rightarrow DataType)$ is the times probability of choosing the data type increases for a given attribute.

Hence, $Lift > 1$ means the data type depends on the attribute and this association rule can be used with a certain level of confidence. The data type suggestion algorithm is given in Fig. 4 below.

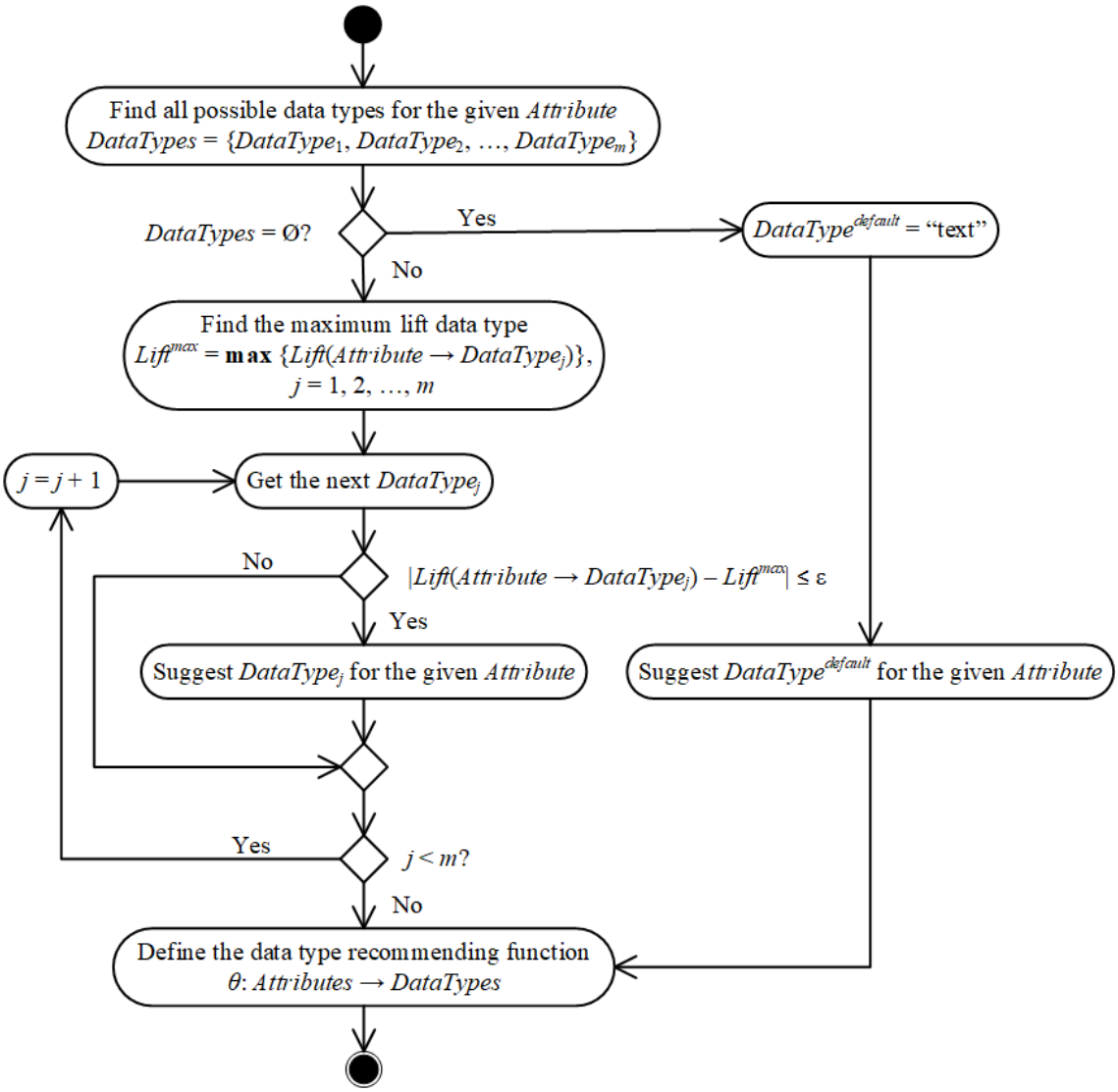


Figure 4: The data type suggestion algorithm based on association rules

According to the proposed algorithm, the data type of maximum lift value or the default data type $DataType^{default}$ (if pairs with a given attribute do not exist) should be suggested:

$$Lift^{\max} = \max_{j=1,2,..,m} \left\{ Lift\left(Attribute \to DataType_j \right) \right\}. \quad (9)$$

The suggested data type should be chosen if the following condition is satisfied:

$$\left| Lift\left(Attribute \to DataType_j \right) - Lift^{\max} \right| \le \varepsilon,\ j = 1,2,\ldots,m, \quad (10)$$

where ε is some tolerance limit or stopping criteria, let us use $\varepsilon = 10^{-6}$.

The "Spider" dataset includes the following data types assigned to different database columns:

$$DataTypes = \{ \text{"number"}, \text{"time"}, \text{"text"}, \text{"boolean"} \}. \quad (11)$$

The default data type is chosen to be a text type, $DataType^{default} = \text{"text"}$.

Table 3 below demonstrates data types suggested for the entity attributes introduced in Table 1.

Table 3
Attribute data types suggested using association rules learning

Attribute	Suggested data type	Support	Confidence	Lift
student_id	number	0.0068	0.97	2.08
full_name	text	0.0002	1.00	2.06
course_number	text	0	-	-
title	text	0.0047	1.00	2.06
number_credits	text	0	-	-
grade	number	0.0006	0.50	1.03

As it is demonstrated in Table 3, attributes "course_number" and "number_credits" are not present in the "Spider" data set. Therefore, an alternative technique should be proposed to augment the used association rules learning algorithm.

3.5. Attribute Data Types Suggestion using Naming Conventions

Let us consider the following sets of keywords that will be used to augment the association rules algorithm by checking how attribute names match naming conventions:

$$\left(K_{num}, K_{time}, K_{bool} \right), \quad (12)$$

where:

- K_{num} is the set of keywords used to recognize number columns;
- K_{time} is the set of keywords used to recognize time columns;
- K_{bool} is the set of keywords used to recognize boolean columns.

Each of the considered keyword sets may contain various tokens used as prefixes or suffixes when naming entity attributes. Some examples of such keywords are given in Table 4.

Table 4
Examples of attribute name prefixes or suffixes

Keywords	Prefix or suffix
K_{num}	"number", "amount", "value", "count", "quantity", etc.
K_{time}	"time", "date", "timestamp", "created", "updated", "modified", "recorded", etc.
K_{bool}	"is", "has", "can", "flag", etc.

Therefore, the mapping between keywords and data types they describe can be formally described as the following function:

$$\varphi : K_{num} \cup K_{time} \cup K_{bool} \to DataTypes. \quad (13)$$

The data type suggestion algorithm based on attribute naming conventions is shown in Fig. 5.

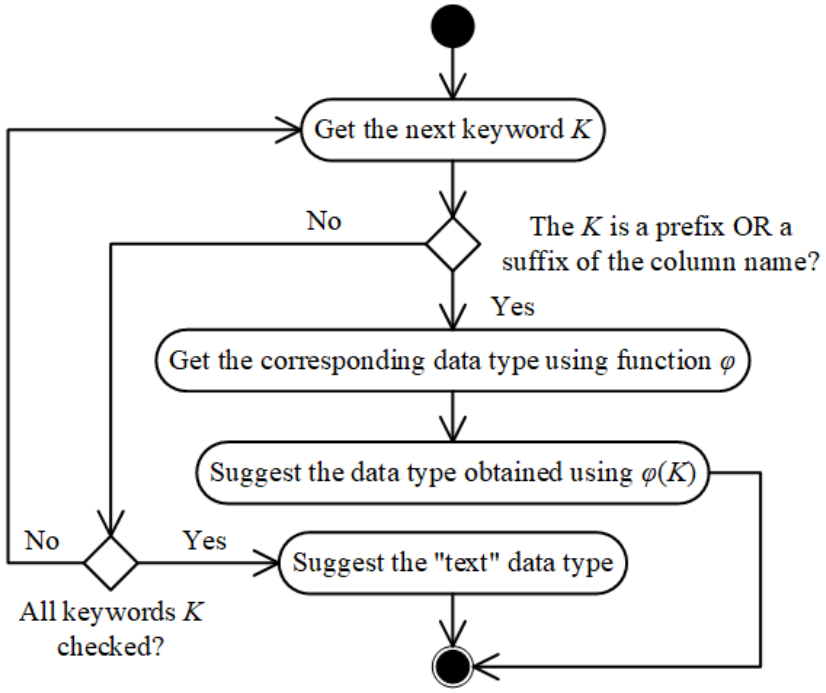


Figure 5: The data type suggestion algorithm based on attribute naming conventions

Hence, the proposed algorithm (Fig, 5) allowed to suggest the “number” data type for attributes “course_number” and “number_credits” previously identified as text attributes (see Table 3).

3.6. Proposed Computational Technology Design

The proposed computational technology includes steps described in sub-sections above and given formally using the following IDEF0 diagram (Fig. 6).

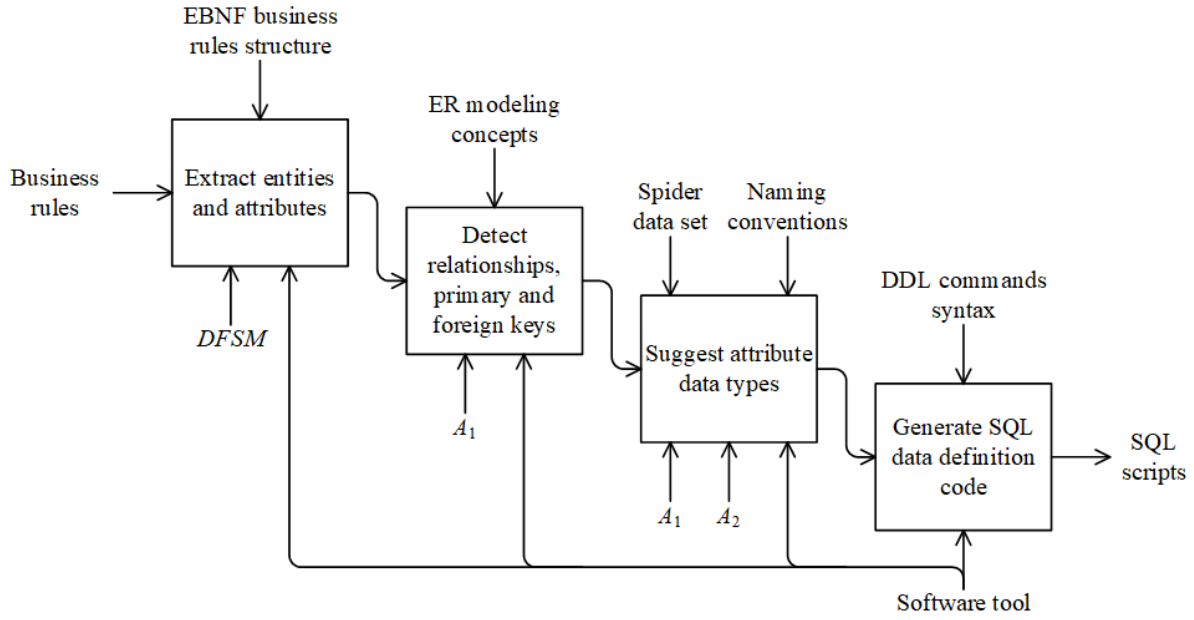


Figure 6: The structural model of proposed computational technology

The final step shown in Fig. 6 includes SQL code generation using the developed software tool.

The proposed computational technology can be formally described as the following tuple:

$$CT = \langle DFSM, AM \rangle, \tag{14}$$

where:

- $DFSM$ is the deterministic finite-state machine introduced in equation (2) and Fig. 1 used for business rules parsing;
- AM is the algorithmic model [18], which consists of the set of interacting algorithms.

The algorithmic model has the following structure:

$$AM = (A = \{A_1, A_2, A_3\}, R \subset A \times A), \tag{15}$$

where:

- $A = \{A_1, A_2, A_3\}$ is the set of interacting algorithms [18];
- A_1 is the relationships detection algorithm;
- A_2 is the attribute data types suggestion algorithm based on association rules;
- A_3 is the attribute data types suggestion algorithm based on naming conventions;
- $R \subset A \times A$ describes connections between algorithms [18].

To perform experiments and validate the proposed computational technology, the software tool for business rules translation into SQL data definition scripts should be developed.

4. Results

4.1. Software Tool Development

The software tool should read a business rules text file as the input, when started by a user, and generate the SQL scripts file as the output. Also users should be able to modify keywords depending on used attributes naming convention (see Table 4).

The UML use case diagram demonstrating the main functional capabilities of the software tool for business rules translation into database creation scripts is demonstrated in Fig. 7 below.

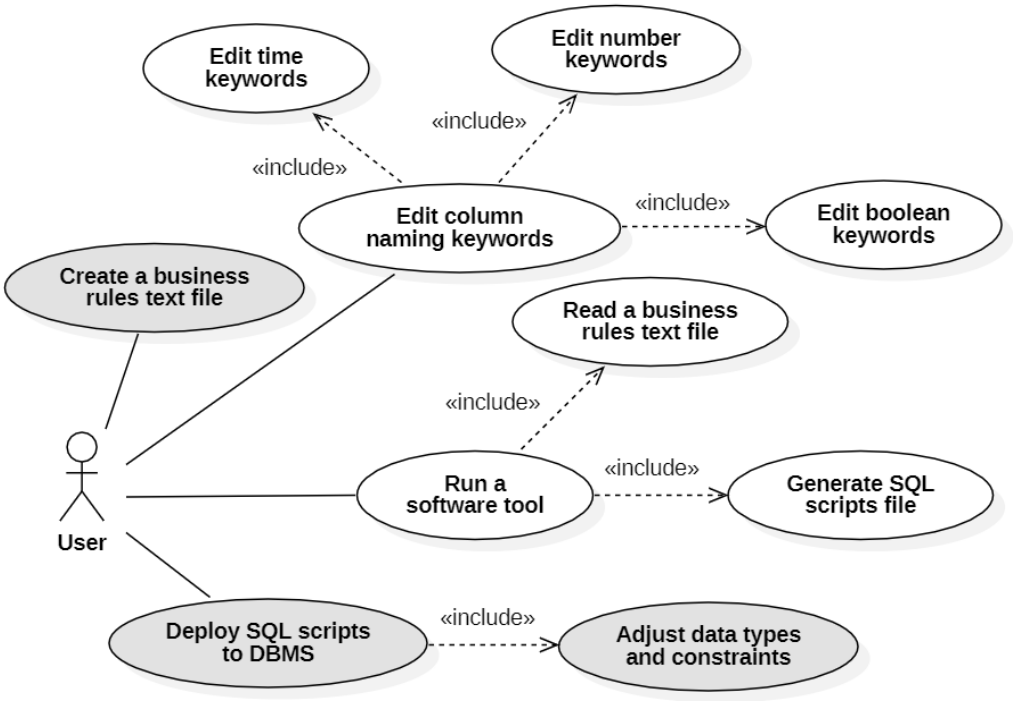


Figure 7: The use case diagram

As it is shown in the use case diagram (Fig. 7) above, some usage scenarios are gray-colored. This means such activities are tightly related to the software tool but executed independently using other software:

- the business rules text file is created using any third-party text editor;
- obtained SQL scripts are adjusted and deployed to DBMS using ant third-party client tool.

The developed computational technology in terms of software components is shown in Fig. 8.

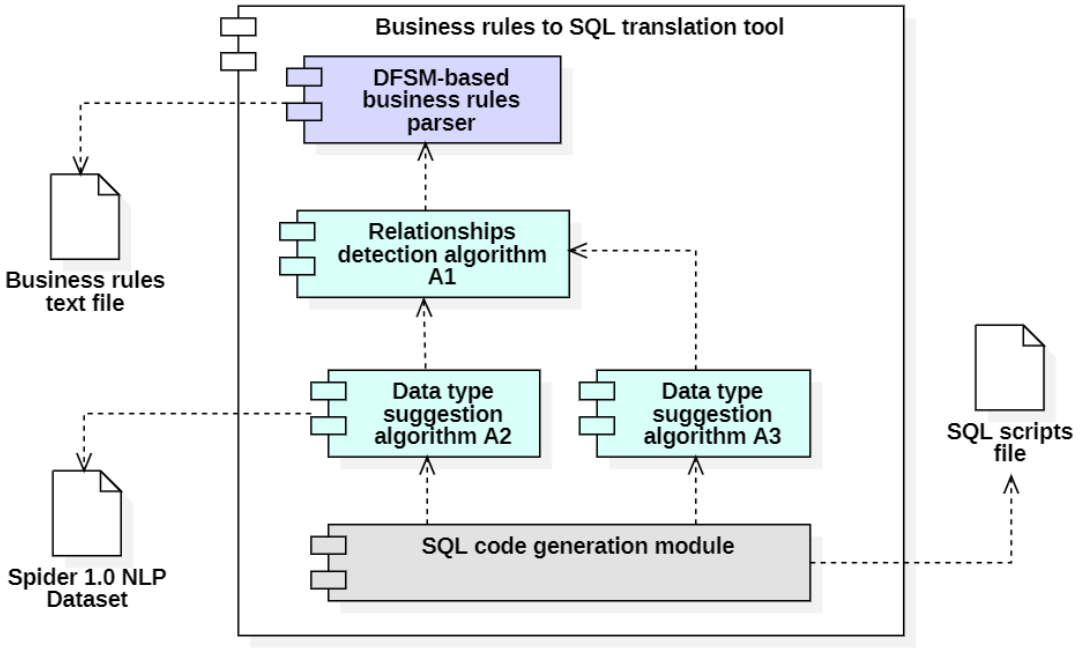


Figure 8: The component diagram

The component diagram above (Fig. 8) demonstrates structural elements mentioned in equations (14) and (15), as well as describes connections between DFSM, algorithms, and SQL code generation software module (for better perception, these components are colored differently).

As for the SQL code generation module, this part of the software tool should solve two tasks:

- build statements according to the SQL syntax;
- substitute generic data types (11) by specific SQL data types.

According to the "DB-Engines Ranking" resource [5], the most popular relational DBMS are:

- Oracle – score: 1247.52;
- MySQL – score: 1195.45;
- Microsoft SQL Server – score: 929.09;
- PostgreSQL – score: 616.50.

Therefore, the corresponding data types used in these DBMS [19] are given in Table 5 below.

Table 5
Specific SQL data types of different DBMS

Data types from the "Spider" data set [16]	Oracle data types	MySQL data types	Microsoft SQL Server data types	PostgreSQL data types
number	NUMBER(18,9)	NUMERIC(18,9)	NUMERIC(18,9)	NUMERIC(18,9)
time	TIMESTAMP	DATETIME	DATETIME	TIMESTAMP
text	CLOB	TEXT	TEXT	TEXT
boolean	NUMBER(1)	BOOLEAN	BIT(1)	BOOLEAN

The software is created using the Python programming language and works as a command-line tool, which takes several arguments, including the name of the business rules text file, the name of the output SQL scripts file, and the name of the target DBMS. It also uses third-party Python modules: Pandas [21] for data processing and Natural Language Toolkit (NLTK) [15] for sentence and word tokenization, and stop-words filtering.

4.2. Software Tool Usage Analysis

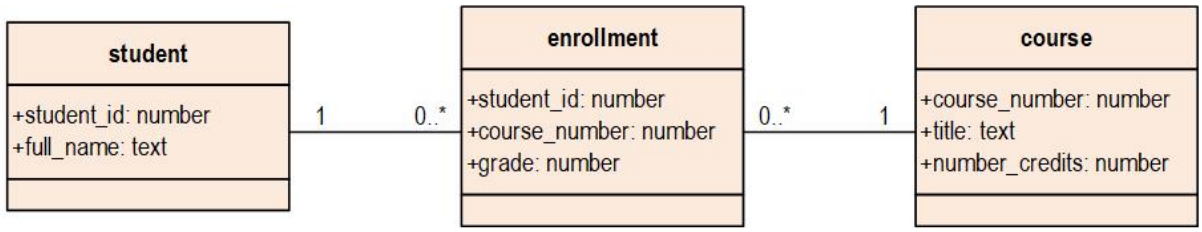
Let us use business rules we introduced in sub-section 2.1 and put them into the input file (Fig. 9).

fsm_br2sql_input.txt

```
1  Each student has a student ID. Each student has a full name. Each course has a course
   number. Each course has a title. Each course has a number of credits. Each enrollment
   has a student ID. Each enrollment has a course number. Each enrollment has a grade.
2
```

Figure 9: The business rules text file

As the result, the software produces the SQL scripts file. The sample output is shown in Fig. 10.




```
1  CREATE TABLE student (
2      student_id NUMERIC(18,9) NOT NULL,
3      full_name TEXT NOT NULL
4  );
5
6  CREATE TABLE course (
7      course_number NUMERIC(18,9) NOT NULL,
8      title TEXT NOT NULL,
9      number_credits NUMERIC(18,9) NOT NULL
10 );
11
12 CREATE TABLE enrollment (
13     student_id NUMERIC(18,9) NOT NULL,
14     course_number NUMERIC(18,9) NOT NULL,
15     grade NUMERIC(18,9) NOT NULL
16 );
```

```
18 ALTER TABLE course
19 ADD PRIMARY KEY (course_number);
20
21 ALTER TABLE enrollment
22 ADD FOREIGN KEY (course_number)
23 REFERENCES course(course_number);
24
25 ALTER TABLE student
26 ADD PRIMARY KEY (student_id);
27
28 ALTER TABLE enrollment
29 ADD FOREIGN KEY (student_id)
30 REFERENCES student(student_id);
```

Figure 10: The sample SQL code generated from business rules

SQL code was produced for various DBMS and tested using "SQL Fiddle" service (Fig. 11) [20].

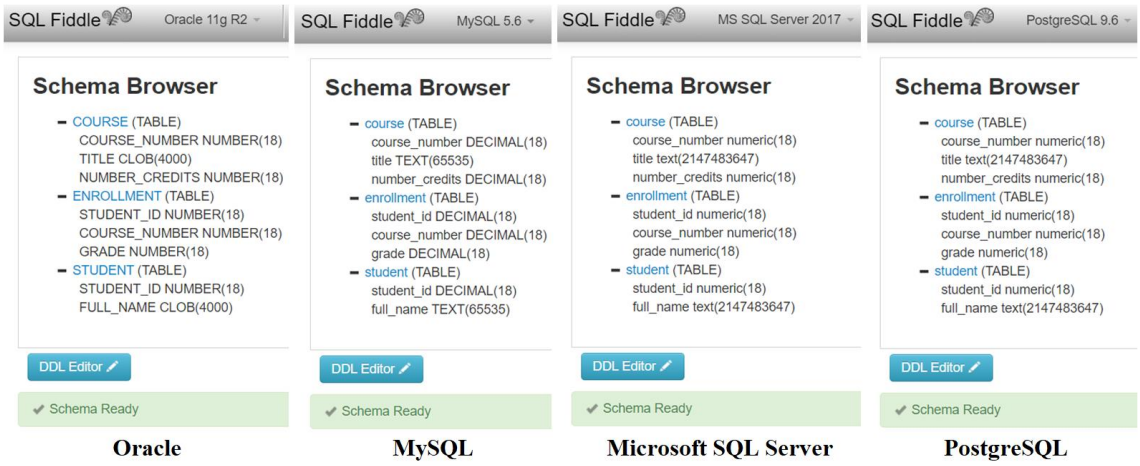


Figure 11: Database schemas successfully created for different DBMS

As it is demonstrated in Fig. 11 above, input business rules (see Fig. 9) were translated into SQL scripts used to successfully create database schemas for Oracle 11g R2, MySQL 5.6, Microsoft SQL

Server 2017, and PostgreSQL 9.6. Obviously, obtained scripts require adjustments of data types and, possibly some other constraints, before being deployed to DBMS servers.

The software tool performance was tested using different amounts of entities and attributes written in business rules (Table 6).

Table 6
Performance testing results

Entities	Attributes	Lines of Code (LOC)	Time, sec
3	8	22	0.04
6	16	44	0.05
15	40	110	0.14
30	80	220	0.28
60	160	440	0.58
150	400	1100	1.57
300	800	2200	3.98
600	1600	4400	6.50
1500	4000	11000	18.32
3000	8000	22000	39.13

These results were also depicted in a chart (Fig. 12) of processing time (in sec.) and SQL scripts size (in LOC) growth as the number of attributes increases.

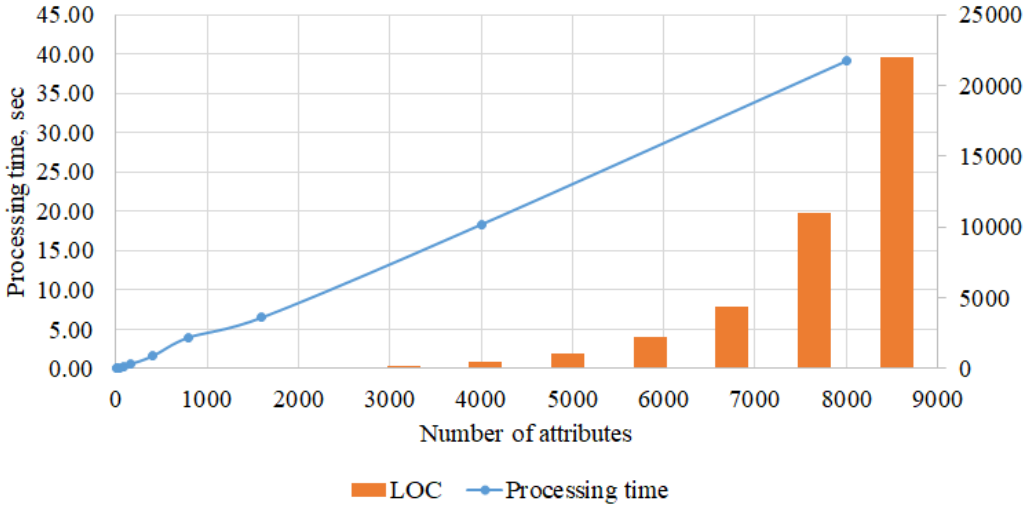


Figure 12: Processing time and LOC growth as the number of attributes increases

The exploratory data analysis measures [22] of performed experiments are given in Table 7 below. It demonstrates minimum, first quartile (Q1), median, third quartile (Q3), and maximum measures of entities, attributes, respective lines of code (LOC), and processing time.

Table 7
Statistical analysis results

Measure	Entities	Attributes	LOC	Time, sec
Minimum	3	8	22	0.04
First quartile	19	50	138	0.17
Median	105	280	770	1.08
Third quartile	525	1400	3850	5.87
Maximum	3000	8000	22000	39.13

The statistical results (see Table 7) demonstrate the minimum processing time of 3 entities and 8 attributes is 0.04 sec. with the produced 22 lines of SQL code. The maximum processing time of 3000 entities and 8000 attributes is 39.13 sec. with the produced 22000 lines of SQL code.

The obtained results (see Table 7) also show the processing time of 25% of business rules is below 0.17 sec. for the input data below 19 entities and 50 attributes, and the output data below 138 lines of SQL code. The processing time of another 25% of business rules is above 5.87 sec. for the input data above 525 entities and 1400 attributes, and the output data above 3850 lines of SQL code.

The remaining 50% of business rules have the median processing time of 1.08 sec. for the input data of 105 entities and 280 attributes, and the output data of 770 lines of SQL code.

The statistical analysis of performed experiments proves the proposed computational technology and software tool are capable of processing large volumes of business rules and generate SQL DDL statements for various popular DBMS in reasonable time. The performance measurements were done on the medium-tier workstation with the Intel Core i5-10210U processing unit of 1.60GHz frequency, 16 GB memory, and 64-bit Windows 10 operating system.

5. Discussion

The software tool is developed as the command-line utility, which takes the input business rules from a text file and generates the output database schema creation scripts according to the proposed computational technology (Fig. 6). The developed software tool is capable of generating SQL scripts compatible with different popular DBMS using various SQL data types supported by these database management systems (see Table 5).

In this study, we considered the most popular relational DBMS: Oracle, Microsoft SQL Server, MySQL, and PostgreSQL [5]. The experimental results (Fig. 9) outline the sample business rules used as the input for the software tool were successfully translated into corresponding DDL code, including table and column definitions, and primary and foreign keys (Fig. 10). The obtained DDL script was successfully tested for database schema generation using the mentioned database engines (Fig. 11). The performance testing of the created software tool demonstrates the capability of solving the large computational problems of SQL code generation from business rules in a reasonable time (Fig. 12).

The proposed technique has several limitations. The current implementation of the proposed idea assumes the usage of only controlled language statements with a restricted structure, defined for fact business rules by equation (1). Also, the detection of relationships between entities could fail if do not mention primary and foreign keys as common entity attributes explicitly. However, it is a common practice to name key attributes as we consider in our approach – among 166 databases present in the "Spider" dataset [16], 163 databases (98%) are using shared names for primary and foreign keys.

6. Conclusion

In this study, we considered the problem of database script generation from natural language text descriptions given as controlled language structures – fact business rules. Fact business rules are brief and precise statements that describe entities of a certain domain. Thus, we formalized their processing using the approach based on a deterministic finite-state machine (Fig. 1). However, extracted entities and attributes are still not sufficient for database design. Therefore, we have proposed an algorithm for relationship detection using the identification of common attributes among different entities (Fig. 2). This algorithm is based on basic concepts of relational databases – primary and foreign keys. To suggest data types for each of the detected attributes, we used the algorithm based on association rules learning (Fig. 4). In addition, we used the other algorithm (Fig. 5) based on naming conventions of attributes to cover those attribute names did not mentioned in the training data set for association rules learning. The proposed computational technology includes the DFSM, proposed algorithms, and the SQL code generation toolkit, and describes the interaction of all of the mentioned components.

In the future, the proposed computational technology should be augmented by algorithms that can detect constraints in business rules (such as null arguments, unique indexes, allowed value ranges for columns, etc.). Also, a web-based software solution should be developed for a better user experience.

7. References

[1] C. Coronel, S. Morris, Database Systems: Design, Implementation, & Management, Cengage Learning, 2018.

[2] F. Tsui, O. Karam, B. Bernal, Essentials of software engineering, Jones & Bartlett Learning, 2022.

[3] D. M. Anderson, Design for manufacturability: How to use concurrent engineering to rapidly develop low-cost, high-quality products for lean production, CRC Press, 2020.

[4] B. Meyer, J.-M. Bruel, S. Ebersold, F. Galinier, A. Naumchev, Towards an Anatomy of Software Requirements, volume 11771 of Lecture Notes in Computer Science, Springer, 2019, pp. 10–40. doi:10.1007/978-3-030-29852-4_2

[5] DB-Engines Ranking, February 2023. URL: https://db-engines.com/en/ranking.

[6] T. Bressoud, D. White, Introduction to Data Systems: Building from Python, Springer Nature, 2020.

[7] M. Javed, Y. Lin, iMER: Iterative process of entity relationship and business process model extraction from the requirements, Information and Software Technology 135 (2021) 106558. doi:10.1016/j.infsof.2021.106558

[8] S. Šuman, S. Candrlic, A. Jakupovic, A Corpus-Based Sentence Classifier for Entity–Relationship Modelling, Electronics 11(6) (2022) 1–22. doi:10.3390/electronics11060889

[9] D. Pieris, M. C. Wijegunesekera, N. G. J. Dias, ER model Partitioning: Towards Trustworthy Automated Systems Development, arXiv (2020). URL: https://arxiv.org/abs/2007.00999. doi:10.48550/arXiv.2007.00999.

[10] S. Hettiarachchi, C. Sugandhika, A. Kathriarachchi, S. Ahangama, G. T. Weerasuriya, A Scenario-based ER Diagram and Query Generation Engine, In 2019 4th International Conference on Information Technology Research (ICITR), IEEE, 2019, pp. 1–5. doi:10.1109/ICITR49409.2019.9407793.

[11] C. Sugandhika, S. Ahangama, Heuristics-Based SQL Query Generation Engine, in: 2021 6th International Conference on Information Technology Research (ICITR), IEEE, 2021, pp. 1–7. doi:10.1109/ICITR54349.2021.9657317.

[12] A. A. Almazroi, L. Abualigah, M. A. Alqarni, E. H. Houssein, A. Q. M. AlHamad, M. A. Elaziz, Class Diagram Generation from Text Requirements: An Application of Natural Language Processing, in: Deep Learning Approaches for Spoken and Natural Language Processing, Springer, 2022, pp. 55–79. doi:10.1007/978-3-030-79778-2_4.

[13] V. Tietz, B. Annighoefer, A formally defined and formally provable EBNF-based constraint language for use in qualifiable software, in: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, 2022, pp. 862–871. doi:10.1145/3550356.3561552.

[14] G. O'Regan, Guide to Discrete Mathematics An Accessible Introduction to the History, Theory, Logic and Applications, Springer International Publishing, 2021.

[15] NLTK – Natural Language Toolkit. URL: https://www.nltk.org/index.html.

[16] Yale University's Spider 1.0 NLP Dataset. URL: https://www.kaggle.com/datasets/jeromeblanchet/yale-universitys-spider-10-nlp-dataset.

[17] T. A. Kumbhare, S. V. Chobe, An overview of association rule mining algorithms, International Journal of Computer Science and Information Technologies 5(1) (2014) 927–930.

[18] V. L. Lysytskyi, Y. Y. Morhun, Development of software for effective enterprise product policy creation, Bulletin of National Technical University KhPI Series System Analysis Control and Information Technologies 21 (2018) 59–64. doi:10.20998/2079-0023.2018.21.11.

[19] B. Brumm, SQL Data Types: Oracle, SQL Server, MySQL, PostgreSQL, 2022. URL: https://www.databasestar.com/sql-data-types/.

[20] SQL Fiddle. URL: http://sqlfiddle.com/.

[21] pandas – Python Data Analysis Library. URL: https://pandas.pydata.org/.

[22] D. T. Bennett, N. M. Scala, P. L. Goethals, Mathematics in Cyber Research, CRC Press, 2022.