An Approach and Software Prototype for Translation of Natural Language Business Rules into Database Structure

Andrii Kopp, Dmytro Orlovskyi and Sergey Orekhov

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

Abstract

In the recent decades data has indeed become one of the most valuable assets for government institutions, private businesses, and individual persons. Nowadays almost any software, from social networks and dating mobile applications to large information systems and analytical services for enterprise management, accumulates, stores, and processes data to solve certain problems in their subject areas. Extremely large data volumes are organized in databases that are used as the baseline for almost all of modern software applications. As the most important components of software systems, databases should be carefully designed, since drawbacks at the stage of requirements elicitation may result in exponential growth of defects fixing costs at testing and maintenance phases. Therefore, this study proposes an approach and software tool to database schema generation from textual requirements also known in database design domain as business rules. This may help database designers to rapidly obtain usable database schemas in order to detect and fix defects as early as possible. Moreover, proposed solution may simplify the database design process, since database creation scripts are generated from business rules directly. Thus, instead of coding all the required statements, engineers are only need to check obtained schema and make certain adjustments to data types, unique attributes, or used naming style. This research considers relational model and relational databases, since they are most widely used nowadays. State-of-the-art analysis is made, proposed approach is described in details, software tool with its brief usage examples is described, conclusions are made, and further research directions are formulated.

Keywords 1

Relational Database Design, Business Rule, Database Schema, Relational Model, Natural Language

1. Introduction

Databases are essential components of almost all modern software systems despite their usage area or architectural complexity, or users demographic. Databases could be considered as computer-based structures that store collections of raw facts, valuable for database users (so called end-user data), and metadata (i.e. data about data) that describes how end-user data is managed. Database management systems (DBMS) are specialized software systems that manage database structures, make collections of data persistent and shareable in a secure way [1]. Originally databases and information systems (IS) that use databases were utilized by enterprises (except some small ones), which had data they needed to store in ways which will be easy to retrieve later [2]. In [2] authors made an example of a ledger of names and addresses of persons or other companies that deal with the enterprise. For small businesses such lists could have been kept on paper, in text files, or spreadsheets. However, tremendous growth of business process complexity, data volumes, and information technology adoption made impossible to store enterprise data without using databases. Nowadays, even simple mobile applications, such as to-do lists, address books, or budget managers, use databases to store users' data in a persistent secure

COLINS-2021: 5th International Conference on Computational Linguistics and Intelligent Systems, April 22–23, 2021, Kharkiv, Ukraine
EMAIL: kopp93@gmail.com (A. Kopp); orlovskyi.dm@gmail.com (D. Orlovskyi); sergey.v.orekhov@gmail.com (S. Orekhov)
ORCID: 0000-0002-3189-5623 (A. Kopp); 0000-0002-8261-2988 (D. Orlovskyi); 0000-0002-5040-5861 (S. Orekhov)

manner. This means that database design is the part of almost any software engineering project and it may require special engineers who have advanced data modeling and database construction skills. It is natural for waterfall or iterative projects, where project teams tend to be large and responsibilities are separated for different specialists or even teams [3]. In pre-agile era, such database design teams were responsible for database design at the early stages of each project and when they had completed tasks for one project, they moved to the next one. But currently agile practices are dominating in software engineering projects, where there are no separate software engineers who exclusively responsible for database design. Database design, implementation, and support in agile projects is done by the same team members who are usually involved in server-side programming [3]. Lack of special training and time to consider the database design more carefully leads to potential design flaws or even mistakes in a database schema. Occurrence of such problems may be prevented by specialized tools that support database design activities when translating gathered requirements into database objects. Hence, in this paper we propose an approach and software tool to translate database design requirements into scripts for database schema generation and its further tuning by responsible project members.

The paper is organized into five sections. Introduction and problem description is given in current section. Section 2 depicts current trends in database design, briefly outlines requirements formulation for database design based on business rules, provides state-of-the-art analysis, and problem statement. Section 3 describes proposed approach to database schema generation from textual business rules and the software design and implementation is outlined in Section 4. In Section 5 conclusion is made and future work in this area is formulated. Sixth section contains only references used in this paper.

2. Related Work

2.1. Current Trends in Relational Database Design

Since the relational model, which established the foundations of databases theory, was introduced by Codd in 1970s, relational DBMS have dominated for several decades. But in the recent decade and half the data management landscape was modified by data structures for which the relational model appeared inefficient. For example, specific systems were designed to handle semi-structured (“raw”) data presented by text and XML (eXtensible Markup Language) files on the one hand, or graph-based “linked” data used by Semantic Web models and languages on the other hand. Also the phenomenon of Big Data refers to data volumes that became so huge that traditional database management systems cannot process them. As a result, so called NoSQL systems appeared as a response to these needs [4].

However, according to the DB-Engines Ranking resource [5], which ranks database management systems with respect to their popularity, the five most popular DBMS are:

- Oracle (relational DBMS, also supports document and graph models).
- MySQL (relational DBMS, also supports document model).
- Microsoft SQL Server (relational DBMS, also supports document and graph models).
- PostgreSQL (relational DBMS, also supports document model).
- MongoDB (document model).

As it is shown, the four most popular database management systems use relational models as their primary engines, while only fifth by popularity DBMS MongoDB uses document store as its primary model. Some of the denoted relational DBMS are proprietary (e.g. Oracle and SQL Server) and some are open-source (e.g. MySQL and PostgreSQL). Nevertheless, these database solutions are not purely relational, since all of the support at least one secondary database model, such as document or graph, or event both [5]. However, the fact that originally relational DBMS are still dominating in February 2021 is quite strong evidence of relational databases actuality and popularity. Even if we take a look at the first 15 database management systems, relational DBMS, such as IBM DB2, SQLite, Microsoft Access, MariaDB, and Azure SQL, still prevail.

In general, relational database design principles did not change much since the relational model was introduced in early 1970s. The primary objects of relational databases to be designed are [6]:

- Database tables.
- Columns within tables (also known as attributes in the relational schema [7]).
- Relationships between tables.

Relational model requires databases to be designed consistent with its principles. Soundness of the database design is defined by following main principles [6]:

- Each table represents exactly one entity in order to avoid redundancy.
- Fields of tables are properly defined and constrained (by domain) in order to prevent errors.

Ranking of DBMS (relational and other kinds) by their popularity [5] is demonstrated in Fig. 1.

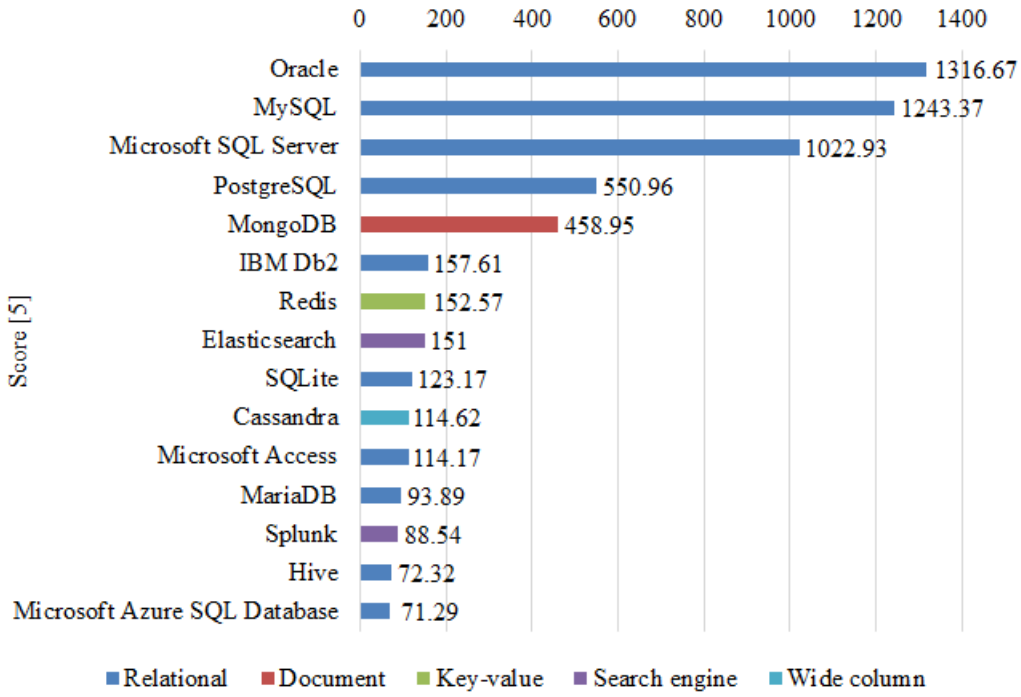

Figure 1: Database management systems ranking [5]

In order to provide relationships between tables, the linking fields should be used for many-to-one or one-to-one relationships, while intermediate linking tables should be used to create many-to-many relationships [6]. Also each table in a relational database has tuples usually referred as rows and fields or attributes usually referred as columns. Tuples and attributes define the two-dimensional structure of a table. While tables represent entities of the subject area, each table column represents corresponding attribute of the entity. Primary keys are used to identify each row of the table uniquely. An attribute or combination of attributes could be used as the primary key. Foreign keys are used to keep references between the tables [7].

The relational database quality is measured using normal forms [1] also referred as normalization levels or degrees [8]. Each level of normalization shows how properly a given database is designed. Normalization is used by database designers to increase data integrity by eliminating anomalies and minimizing data redundancy. The three types of anomalies are possible: modification, insertion, and deletion anomalies [8].

The objective of normalization is to ensure that a database schema is at least in third normal form (3NF), even though high-level normal forms exist [1]. On practice the 3NF might seem as sufficient to ensure the data integrity and consistency. In [9] Carpenter has explained to which normal form a database designer should aspire. According to this paper, designer should reach the 4NF in order to avoid anomalies that might still exist at lower levels of normalization. However, a high granularity of relations, obtained after normalization, may result in efficiency and performance issues, increase of software complexity, and lack of convenience for users and database administrators [9]. Hence, the tradeoff between data vulnerability to anomalies and system performance should be reached. In [10] authors state that databases not normalized to the 3NF will have anomalies and data consistency will not be guaranteed. Author of [11] addresses the 3NF as the generally regarded industry standard for relational databases. It is a common practice to recommend decomposition of databases to the third normal form, since it is considered as lossless decomposition, it preserves functional dependencies, and it is resistant to anomalies [11].

2.2. Business Rules in Database Design

In database design business rules are used as sources for correct discovery of entities, attributes, constraints, and relationships. Business rules are brief, precise, and unambiguous textual descriptions of policies, processes, and principles within a certain organization. The main sources of business rules are people and documentation within organization: managers of different levels, company policies, or process manuals. Also business rules could be elicited by directly interviewing end users that will use a database and an IS under design [1]. From the IS design perspective, business rules are formulated as statements that define or constraint some aspects of organizational activities. Simplified taxonomy of business rules includes six categories [12]:

- Facts. Statements that define entities and relationships within data models.
- Constraints. Statements that restrict actions, which IS or its users are allowed to perform.
- Action enablers. Rules that trigger some activities if certain conditions are true.
- Inferences. Rules that can be used to create new facts from already existing facts, often such business rules are written using "if-then" statements (as well as action enablers).
- Computations. Specific equations or algorithms that transform existing numerical values into new numerical values.
- Terms. Words, abbreviations, and phrases valuable to a business. Often merged with facts.

In contrast to Wiegers [12], whose book is focused on software requirements elicitation in general, luminaries of database design Coronel and Morris [1] define database business rules as definitions of data model components (entities, attributes, relationships, and constraints). Business rules by Coronel and Morris [1] are close to "facts" business rules by the taxonomy presented in [12]. Therefore, in next sections business rules will be referred exactly as definitions of entities, attributes, relationships, and constraints.

2.3. State-of-the-Art and Problem Statement

Translation of business rules into data model components and then into database schema requires involvement of engineers with special skills and certain experience. Database designers need to check all the requirements, gathered by business analysts and presented in the form of business rules [1, 12], translate such requirements into entity-relationship models, and implement data models using DBMS. Described activity is indeed the bottleneck of all database design process, since cost of errors occurred and undetected at the requirements stage may increase dramatically at further stages [13]. To prevent such risks, agile software development methodologies ensure improvement of product quality through rapid deliveries of usable software components at each stage of the lifecycle [14]. Existence of usable prototype is vital for quality assurance, so the earlier the database schema (or at least the data model) can be obtained from business rules, the earlier the defects can be detected [15].

Analysis of the state-of-the-art has shown only several studies devoted to business rules translation into a database structure. In [16], Bajwa et al. proposed an approach of translating natural language specifications to SBVR (Semantic Business Vocabulary and Rules) business rules. They have adopted SBVR, which is a standard introduced by OMG (Object Management Group) consortium to represent informal specifications, which usually captured in natural languages, in the formal logic for machine-processing [16]. Translation of SBVR business rules into SQL (Structured Query Language) queries is described in [17]. However, this research paper is focused on DML (Data Manipulation Language) operations, such as SELECT, UPDATE, INSERT, and DELETE, on a relational schema [17]. Most recent paper by Kate et al. [18] considers natural language query to SQL query translation but only for data retrieval. As for translation of textual descriptions into DDL (Data Definition Language) scripts, such studies have not been discovered yet. Our previous study [19] only introduced considered idea.

Thus, considered research direction is relevant and might be elaborated. Research objective of this paper includes a process of business rules translation into a database schema. Research subject is the approach and software tool for textual business rules translation into DDL scripts to create a database on DBMS server. This study aims to improve the database design process by decreasing of errors that may occur when business rules are translated into data models and then into database schemas.

3. Database Schema Generation from Textual Business Rules

3.1. Business Rules to Relational Model Translation

3.1.1. Business Rules Formalization

In this study business rules are considered as verbal definitions of main database objects, including entities, attributes, relationships, and constraints. According to the business rules taxonomy given in [12], we focus on fact business rules. At the same time it is recommended to use "well-structured and carefully written natural language" [12]. Therefore, fact business rules that describe database entities and relationships between such entities could be described using the following tuple:

$$Rel = \langle Adj, E_{parent}, Descr, Card, E_{child} \rangle, \quad (1)$$

where:

- Adj is the adjective in the beginning of each business rule that describes relationships among database entities, e.g. "Each student is enrolled to many courses", $Adj \in \{"each", "some"\}$;
- E_{parent} is the identifier of a database entity from which relationship is directed to another one database entity (so-called "parent" entity);
- $Descr$ is the description of a relationship between two database entities, e.g. "is assigned to", "is related to" etc.;
- $Card$ is the relationship cardinality between two database entities, $Card \in \{"one", "many"\}$;
- E_{child} is the identifier of database entity to which the relationship is directed from another one database entity (so-called "child" entity).

As it is demonstrated below, sample fact business rule "Each student is enrolled to many courses" could be described formally as: $Adj = "each"$, $E_{parent} = "student"$, $Descr = "is\ enrolled\ to"$, $Card = "many"$, and $E_{child} = "course"$ (plural database entity identifiers extracted from business rules should be singularized). Therefore, corresponding business rule takes the following form:

$$Rel_{Student-Course} = \langle "each", "student", "is\ enrolled\ to", "many", "course" \rangle. \quad (2)$$

Fact business rules are used not only to describe relationships between database entities, but also to describe attributes of database entities. Such business rules could be described using the following tuple:

$$Ent = \langle Adj, E, Pred, Attr \rangle, \quad (3)$$

where:

- Adj is the adjective in the beginning of each business rule that describes attributes of database entities, e.g. "Each student has full name, birth date, enrollment date", $Adj = "each"$;
- E is the identifier of database entity;
- $Pred$ is the predicate, $Pred = "has"$;
- $Attr$ is the set of attributes that describe a database entity, $Attr = \{attr_i | i = \overline{1,n}\}$, where n stands for the number of attributes in an entity.

As it is demonstrated below, sample fact business rule "Each student has full name, birth date, enrollment date" could be described formally as: $Adj = "each"$, $E = "student"$, $Pred = "has"$, and $Attr = \{"full\ name", "birth\ date", "enrollment\ date"\}$. Therefore, a business rule is written as:

$$Ent_{Student} = \langle "each", "student", "has", \{"full\ name", "birth\ date", "enrollment\ date"\} \rangle. \quad (4)$$

Next subsection describes how such business rules (1) and (2) are extracted from textual domain descriptions provided by business analysts or other stakeholders.

3.1.2. Business Rules Extraction

In order to demonstrate syntax of business rules provided as text, the Extended Backus-Naur Form (EBNF) is used [20]. Therefore, fact business rules that describe database entities and relationships between such entities should have the following structure:

$$< relationship\_business\_rule > ::= \{each|some\} < parent\_entity > \{is < relation >\} \{one|many\} < child\_entity >. \quad (5)$$

Fact business rules that describe attributes of database entities should have the following structure, which is also described using the EBNF [20]:

$$< entity\_business\_rule >::= \{each\} < entity > \{has\} \{< attribute >, \ldots\}. \quad (6)$$

We assume that textual description passed as input contains business rules provided as sentences, so we could split input text into the set of sentences, and then process each sentence as a standalone business rule. Formally input text that may contain business rules could be described using following equation:

$$BR_{Text} = \{BR_j | j = \overline{1,m}\}, \quad (7)$$

where:

- BR_j is the one of sentences that form textual domain description, and which may be parsed as certain business rule;
- m is the number of sentences (potential business rules) provided in considered textual domain description (7).

Business rules extraction procedure includes following steps:

1. Preparation step.

In order to extract sentences BR_j, $j = \overline{1,m}$ that may turn out to be business rules, it is required to split input text BR_{Text} using "." (dot) character as the separator. Obtained collection (an array or a list depending on the software implementation in future) then need to be processed in order to:

- Avoid sentences (string values) of zero length.
- Trim sentences (string values), i.e. remove leading and trailing spaces.

2. Identification step.

Each of extracted sentences BR_j, $j = \overline{1,m}$ must be brought to the one of two fact business rule types: ones that describe relationships, or ones that describe entity attributes. This could be done using regular expressions based on introduced syntax descriptions (5) and (6):

- The regular expression to parse business rules that describe database entities and relationships between such entities has the following form:

$$Rx_{Rel} = "(each|some)\s+(.+)\s+((is).+)\s+(one|many)\s+(.+)"; \quad (8)$$

- The regular expression to parse business rules that describe attributes of database entities has the following form:

$$Rx_{Ent} = "(each)\s+(.+)(has)\s+(.+)". \quad (9)$$

If certain sentence BR_j, $j = \overline{1,m}$ does not match to any of these regular expression (8) and (9), this sentence should be excluded from consideration as of potential business rule.

3. Relational mapping step.

When certain sentence BR_j, $j = \overline{1,m}$ matches to the regular expression Rx_{Rel}, there are following tokens that we may extract (see Fig. 2).

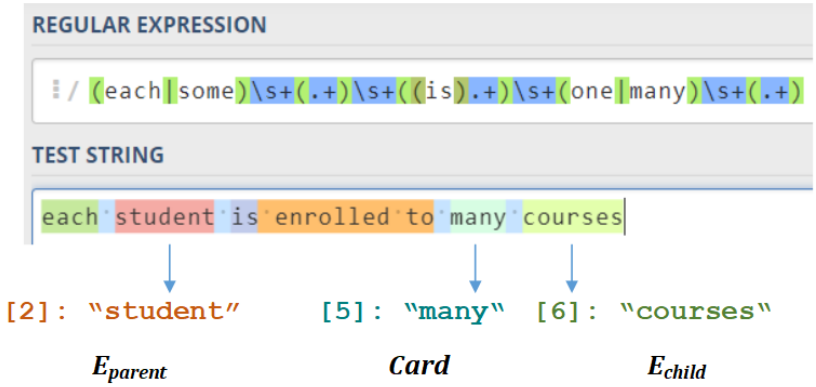

Figure 2: Example of business rule matching to the regular expression Rx_{Rel}

All parsed tokens that correspond to "parent" entity E_{parent}, relationship cardinality $Card$, and "child" entity E_{child} should be trimmed (there should be removed leading and trailing spaces for each

of the extracted tokens), and any sequences of spaces presented in such tokens must be replaced with underscores "_" in order to avoid syntax errors when translating database structure to DDL language commands. Before sentences are parsed they should be translated to lower case.

Here it is necessary to introduce the following equation, which denotes that each table Tab maps database entity identifiers to structural descriptions of database tables:

$$Tab: E \to \langle PK, FK, Cols \rangle, \tag{10}$$

where:

- E is the identifier of database entity to which certain database table corresponds;
- PK is the primary key columns of database table Tab;
- FK is the set of foreign key columns and table references of database table Tab;
- $Cols$ is the set of remaining non-key column names derived from the attributes set $Attr$ (3).

Relational mapping of business rules that describe database entities and relationships between such entities includes the following stages:

- If mapping Tab (10) does not exist for the "parent" entity E_{parent}, then it should be provided as following:

$$Tab(E_{parent}) \to \langle E_{parent} + "\_id", \emptyset, \emptyset \rangle. \tag{11}$$

As it is demonstrated in (11), there should be used concatenation of "parent" entity identifier and "_id" suffix as the primary key. For example, for "student" entity there should be created PK named "student_id". Obtained structure could be represented using the fragment of entity-relationship model shown in Fig. 3.

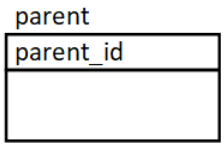

Figure 3: Visual representation of "parent" table mapping provided in (11)

- If mapping Tab (10) does not exist for the "child" entity E_{child}, then it should be provided as following:

$$Tab(E_{child}) \to \langle "", \emptyset, \emptyset \rangle. \tag{12}$$

As it is demonstrated in (11), there should be used empty primary key until relationship cardinality is checked. Obtained structure could be represented using the fragment of entity-relationship model shown in Fig. 4.

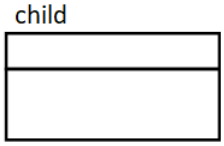

Figure 4: Visual representation of "child" table mapping provided in (12)

- If relationship cardinality $Card$ is equal to "one", then for the mapping $Tab(E_{child})$ primary key column PK should be set to $E_{parent} + "\_id"$ (in order to implement one-to-one identifying relationship between two entities), while the FK set should be appended with the following pair of foreign key column and table reference: $\langle E_{parent} + "\_id", E_{parent} \rangle$. Obtained relationship could be represented using the fragment of entity-relationship model shown in Fig. 5.

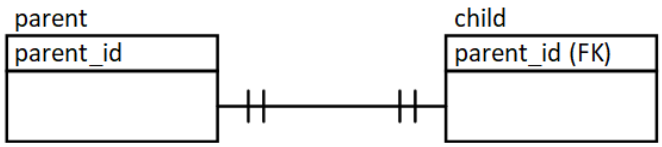

Figure 5: Visual representation of one-to-one relationship between "parent" and "child" tables

- Otherwise, if cardinality $Card$ is equal to “many”, then for the mapping $Tab(E_{child})$ primary key column PK should be set to $E_{child} + \text{"\_}id\text{"}$ (since one-to-many relationship is non-identifying), while the FK set should be appended with the same pair of $\langle E_{parent} + \text{"\_}id\text{"}, E_{parent} \rangle$. Obtained relationship could be represented using the fragment of entity-relationship model shown in Fig. 6.

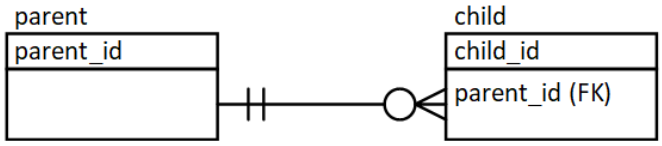

Figure 6: Visual representation of one-to-many relationship between “parent” and “child” tables

When certain sentence BR_j, $j = \overline{1,m}$ matches to the regular expression Rx_{Ent}, there are following tokens that we may extract (see Fig. 7).

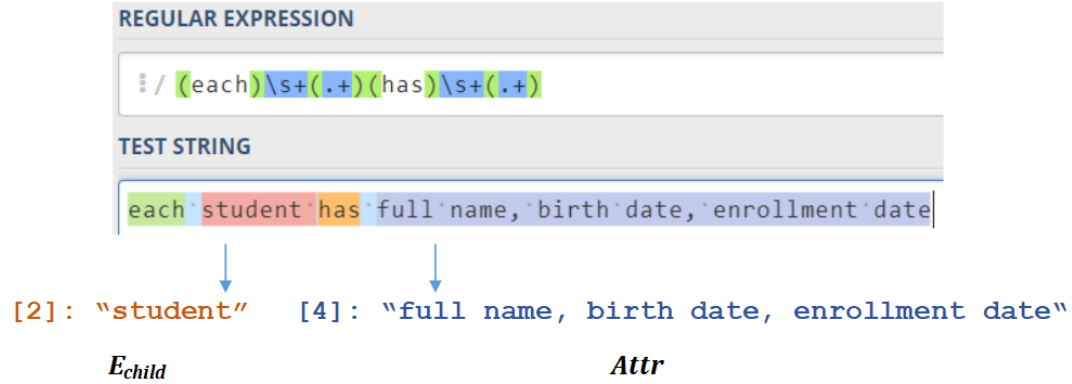

Figure 7: Example of business rule matching to the regular expression Rx_{Ent}

All extracted tokens, which correspond to the entity E and attributes set $Attr$, should be trimmed (leading and trailing spaces should be removed for each of the extracted tokens). The substring that corresponds to attributes set $Attr$ should be separated using comma “,” in order obtain collection of attribute names. Also all the sequences of spaces presented in extracted tokens (entity and attribute names) are replaced with underscores “_” in order to avoid syntax errors when translating database structure to DDL language commands. Before sentences are parsed they should be translated to lower case respectively.

Relational mapping of business rules that describe attributes of database entities includes the next stages:

- In case if mapping Tab (10) does not exist for considered entity E, then it should be provided as following:

$$Tab(E) \to \langle E + \text{"\_}id\text{"}, \emptyset, \{attr_i | i = \overline{1,n}\} \rangle. \quad (13)$$

- Otherwise, existing mapping $Tab(E)$ should be appended by adding extracted and processed set of attributes $\{attr_i | i = \overline{1,n}\}$ as column names.

Obtained structure could be shown using the fragment of entity-relationship model in Fig. 8.

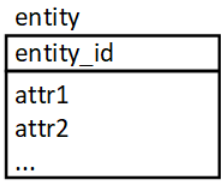

Figure 8: Visual representation of entity mapping provided in (13)

The example of relational mapping produced by the described business rules extraction procedure is demonstrated in Fig. 9. Obviously demonstrated example (Fig. 9) is not perfect and vulnerable for

anomalies, however it is shown only to display how business rule statements are translated into entity-relationship structures. However, compliance of generated relational structures to the 3NF depends only on business rules granularity and elaboration: formulated business rules should be free of partial and transitive dependencies, multi-valued attributes, and multi-valued dependencies [9].

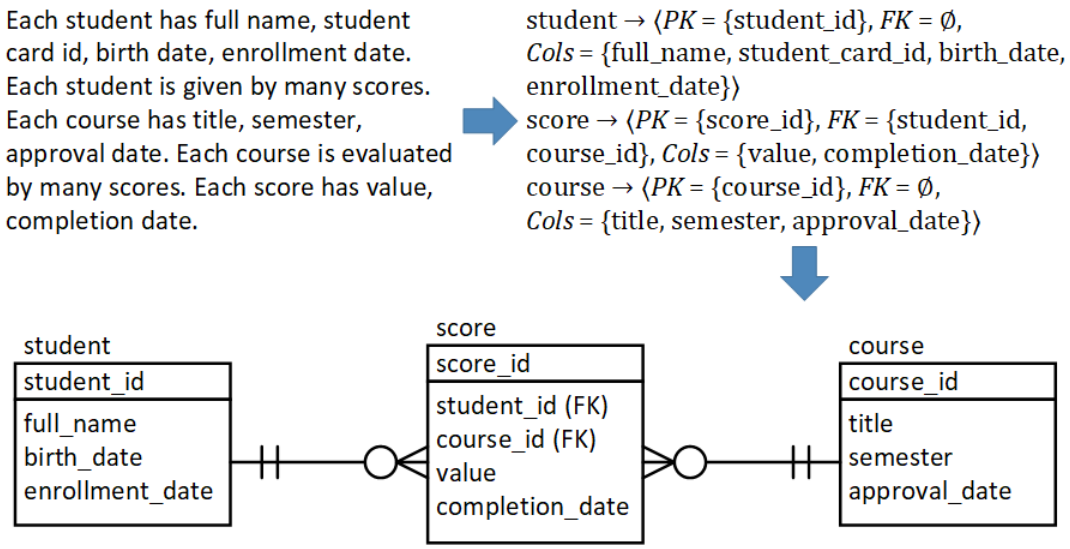

Figure 9: Example of relational mapping produced by business rules extraction procedure

Demonstrated relational model (Fig. 9) contains "full_name" attribute of "student" entity, which could be decomposed into at least two atomic attributes to store first name and last name respectively. Another drawback of demonstrated approach, is that for now it is limited to support only one-to-many or one-to-one relationships. Also, in the outlined example, many-to-many relationship was achieved by using two one-to-many relationships from both "sides" of the "score" entity. This technique may mislead business analysts or other users who are responsible for business rules description but do not have enough training and experience in relational database modeling.

3.2. Relational Model to Database Scripts Translation

3.2.1. Database Scripts Generation

Proposed business rules extraction procedure produces structures Tab (10), which map database entity identifiers to structural descriptions of database tables. These structures describe meta-data of a future database and, hence, could be translated into DDL statements.

At first, for each mapping should be generated CREATE TABLE scripts using the following rules:

- Database entity name E is translated into a table name.
- The primary key PK attribute is translated into the INTEGER column. Since primary keys are artificial identifiers (e.g. 1,2,3, …), corresponding columns should have auto-increment properties.
- Those foreign keys FK, which are not included in primary key, are translated into INTEGER columns as well.
- Attributes $Cols$ are translated into non-key columns. At this point, the black-box function that maps attributes to three generic domains (date and time, numerical, and text data) is introduced:

$$domain: attr_i \to \{DateTime, Decimal, Varchar\}, i = \overline{1,n}. \quad (14)$$

Obviously suggested domains may be adjusted by the database designer if necessary. Proposals for the data types, which could be chosen for considered domains (14) are outlined in next subsection. As it was outlined in 3.1.2, primary keys are generated automatically and contain auto-increment integer values. But there may be present unique alternate keys, which identification is considered in 3.2.3.

The example of CREATE TABLE scripts generation, shown in Fig. 10, is based on the example of relational mapping demonstrated in Fig. 9.

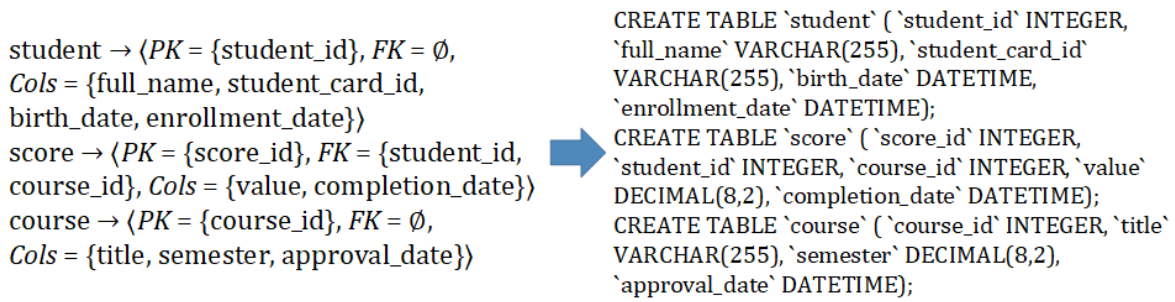

Figure 10: Example of CREATE TABLE scripts generation to create tables

After database tables are declared using CREATE TABLE commands, it is necessary to establish keys and relationships between the tables. This means adding primary key constraints and foreign key references to "child" tables. This can be done using special ALTER TABLE scripts, which examples for the considered topic are shown in Fig. 11.

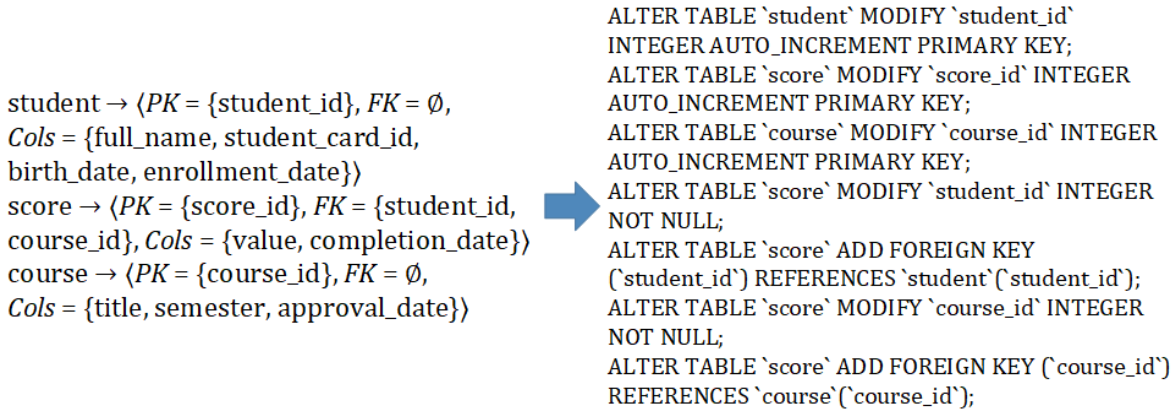

Figure 11: Example of ALTER TABLE scripts generation to create foreign keys

When mapping the relational model into DDL statements, primary keys should be implemented as auto-increment integer columns in order to simplify their processing in a database. Obviously primary attributes are all have NOT NULL property, as well as the foreign key attributes should have NOT NULL property in order to provide the strong relational integrity. Auto-increment keys and not-null fields are the essentials of relational databases and supported by all well-known relational DBMS.

By default, primary keys, as well as the corresponding foreign keys, are all artificial (entity name concatenated with "_id" suffix as it was demonstrated above), however there are also alternate keys possible for some domain-specific attributes. These alternate keys are also supposed to be not-null, since they are used to represent real-world identifiers. Detection of these keys, as well as the domains for table columns, is outlined in the next subsections.

3.2.2. Column Domains Suggestion

Suggestion of domains or data types using only column names is a non-trivial goal, which could be accomplished using naming conventions and commonly used attribute names that belong to specific domains. For this purpose, we have used the schema.org vocabulary [21], which declares four generic domains:

- DateTime.
- Number.
- Text.
- Boolean.

For this domains should be introduced a vocabulary of categorized attribute titles, which could be then matched to attribute titles of database entities when DDL statements are generated.

Frequencies of attribute title occurrences among the vocabulary of titles categorized by considered domains are calculated and the Naïve Bayes approach [22] could be used to suggest domains, since it is fast enough for real-time multiclass predictions and does not need large volumes of training data:

$$c = \arg\max_{c \in C} P(c|d), \quad (15)$$

where:

- C is the set of domains used as classes, $C = \{DateTime, Number, Text, Boolean\}$;
- $P(c|d)$ is the number of titles, which belong to the domain $c \in C$, matched to the attribute d.

However, when using (15) there may occur situations when frequency values $P(DateTime|d)$ and $P(Number|d)$ are equal for respective domains. In such case, the Text domain could be suggested in order to provide flexibility of data storage, and as the one more reason for database tuning. Obviously suggested domains, as well as tables, columns, keys, and relations, do not show final database design and could or even must be tuned with respect to the considered subject area.

Therefore, the vocabulary of attribute titles categorized by domains should be extended over time. Hence, the following human-computer interaction procedure could be introduced (Fig. 12).

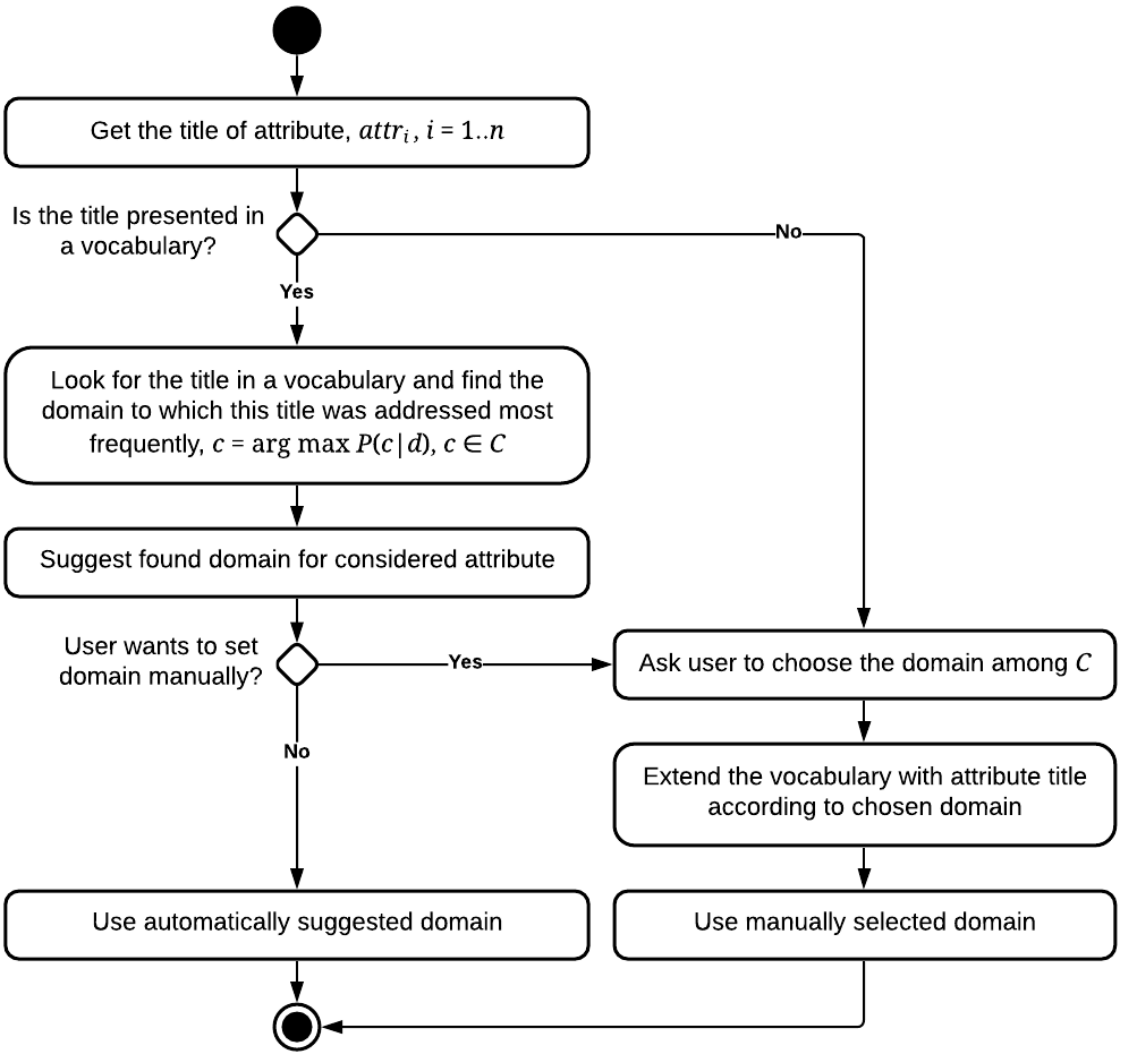

Figure 12: Procedure of column domains suggestion

Modern enterprise-level DBMS support various options of data types that correspond to suggested domains. For example, DATETIME for the DateTime domain, DECIMAL for the Number domain, VARCHAR for the Text domain, and TINYINT(1) for the Boolean domain. Numbers of digits (in total and after the decimal point) for DECIMAL data type and characters for VARCHAR data type should be manually selected by the database designer or database administrator in order to achieve better performance and provide sufficient space for stored data.

3.2.3. Alternate Keys Suggestion

As it was outlined above, the detection of possible alternate keys is necessary in order to assure uniqueness of certain values like social-security number (SSN), vehicle identification number (VIN), personal identification (ID) card number, bank account number etc.

Similarly to the column domains detection procedure, the vocabulary of column names that could be used as UNIQUE indexes should be introduced. Frequencies of attribute title occurrences among the vocabulary of possible alternate key titles are calculated and the logistic activation function [23] could be used to formalize suggestion of UNIQUE indexes:

$$p(x) = \frac{1}{1 + e^{-x}}, \tag{16}$$

where x is the frequency of attribute title occurrences among the vocabulary of alternate key titles.

We use logistic or sigmoid activation function (16), since it comes along with the threshold value suitable for binary classification. For example, if a value obtained using the logistic model is greater than 0.5, then such column might be used as unique index [23]. The higher returned value is, the more chances this is indeed an alternate key. In further research there may be considered special threshold values to filter alternate key suggestions, as well as more advanced classification methods. Also the vocabulary may be extended over time when some specific subject areas are considered. Hence, the following human-computer interaction procedure could be introduced (Fig. 13).

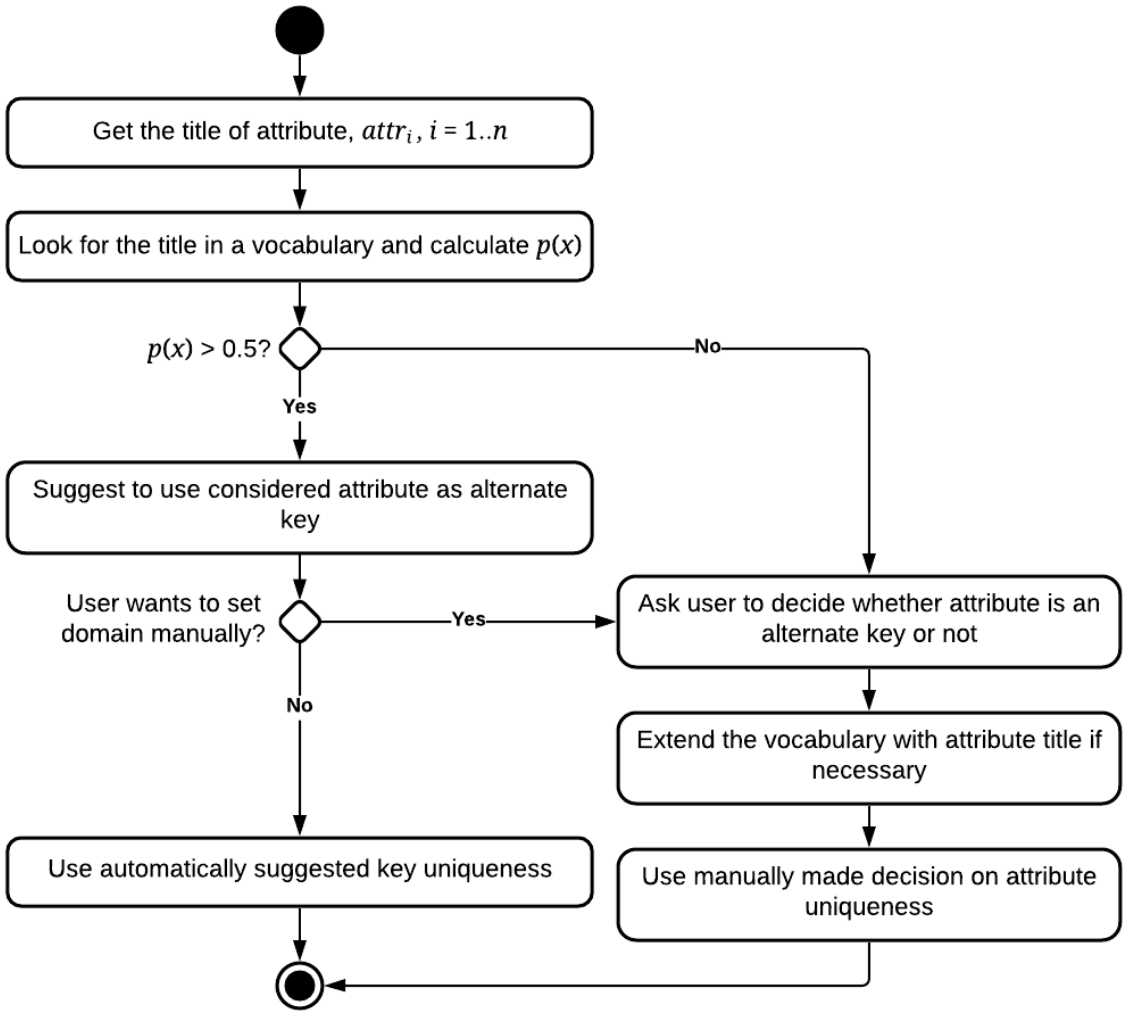

Figure 13: Procedure of alternate keys suggestion

As well as the column domains, alternate keys could or sometimes must be manually tuned before a database is deployed to the server. Detected alternate keys could be implemented as unique indexes,

which are supported by all modern relational DBMS. This could be achieved using special ALTER TABLE scripts, which examples for the considered topic are demonstrated in Fig. 14.

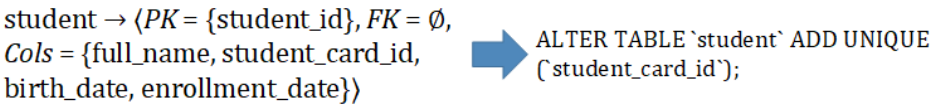

Figure 14: Examples of ALTER TABLE scripts generation to create unique indexes

Demonstrated statement (see Fig. 13) may be executed in addition after DDL statements sufficient to create a database were generated using procedures proposed in subsection 3.2.1. Also it should be noticed that outlined procedures use MySQL-based dialect of DDL scripts that may not work properly in other DBMS. However, in the software implementation it is planned to add multi-system support of generated DDL scripts.

4. Software Design and Implementation

4.1. Software Tool Prototype

The software tool is supposed to be used by business analyst and database designer roles. Business analysts should have possibilities to create projects, setup vocabularies used to detect column domains (see Fig. 12) and alternate keys (see Fig. 13), and add business rules for database schema generation. If necessary, business analysts should be able to update or delete any vocabulary elements or business rules in projects they are participating.

Database designers, as well as business analysts, should be able to review projects, however, they also should be able to obtain DDL scripts generated from formulated business rules. Scripts should be configurable (e.g. to include or exclude checks for already existing database and tables with the same names etc.). Also it is supposed to provide database designers with the possibility to execute obtained DDL scripts after connecting to a database management system server. Main features of the software tool are shown on UML (Unified Modeling Language) use case diagram in Fig. 15.

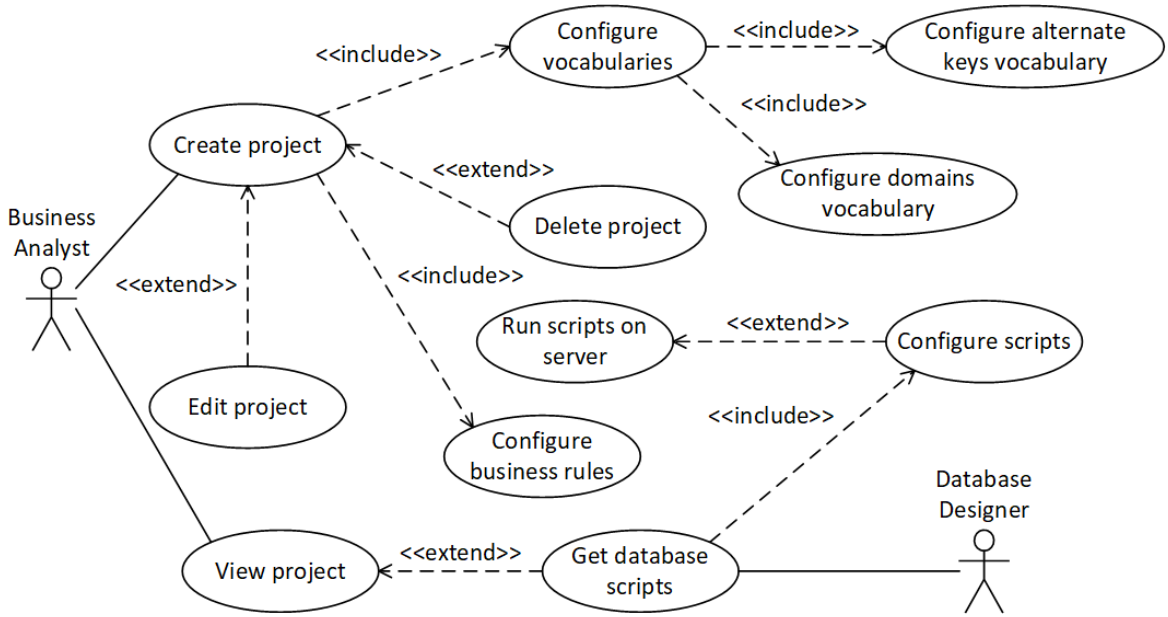

Figure 15: Planned use case scenarios

At this moment the software is implemented only as the prototype, which allows users to translate given business rules, provided as text, into DDL scripts to create a database. The software prototype is implemented as a server-less web application: there is a single HTML (HyperText Markup Language) home page, which uses Bootstrap front-end library to create the responsive user interface and jQuery

library to simplify DOM (Document Object Model) operations. Described business rules processing and translation into DDL statements is also implemented using JavaScript. Several objects, which are used to detect attribute domains and alternate keys, translate business rules into a relational structure, and then into DDL statements that describe the database schema, were developed. System architecture of the software prototype is shown on UML deployment diagram demonstrated in Fig. 16.

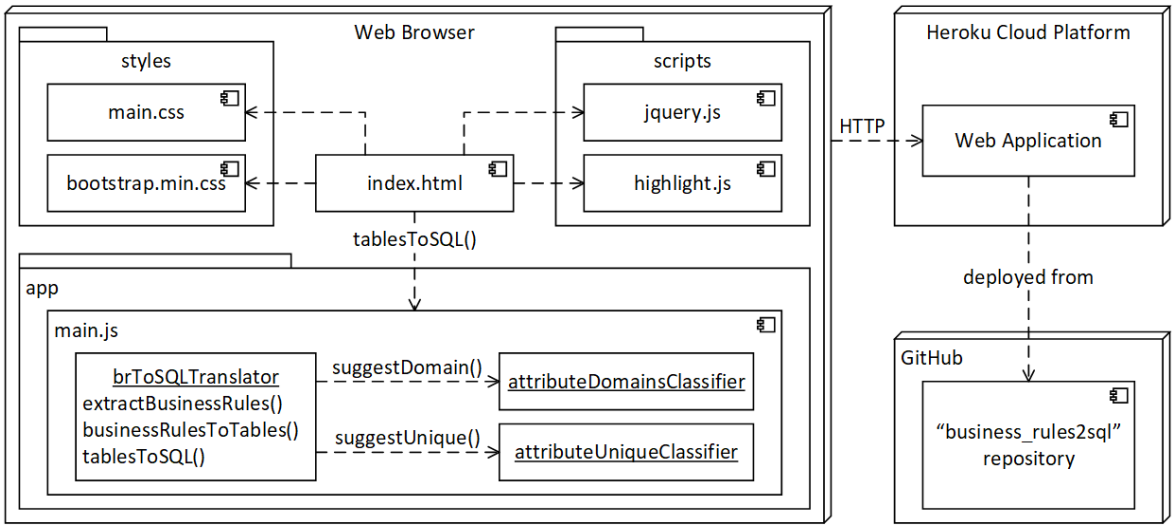

Figure 16: System architecture of the software prototype

As it is demonstrated in Fig. 15, the source code is available in GitHub repository [24] also used as the source for application deployment into the Heroku cloud-based platform. Organized development pipeline allows to immediately deploy recent changes of the software, which helps to improve internal communication between developers and testers, and receive latest feedbacks from stakeholders.

4.2. Analysis of Software Prototype Usage Results

The software prototype is now accessible online for experiments [25]. Since its functions now are limited only to business rules input and DDL scripts output with minimum configurations (users only may choose whether CREATE DATABASE statement and should be included, as well as to choose if DROP statements are necessary before creation of a database and its tables), it has been implemented as the single-page web application that fits to a single screen. A home page is demonstrated in Fig. 17.

The home page includes several user interface elements:

- Text area where users may put their business rules, which describe certain subject area.
- Buttons, which are used to clear the input text area or translate given business rules into DDL scripts respectively.
- Source code area, where DDL statements, translated from given business rules, are shown.
- Text input to provide that database name.
- Check boxes for optional settings (whether CREATE DATABASE, DROP DATABASE, and DROP TABLE should be included).
- Radio button to select the SQL dialect used to generate DDL statements (for now it supports only MySQL dialect).
- Button to copy generated DDL statements to clipboard in order to simplify scripts transfer to a DBMS client.

In order to demonstrate the software usage example (see Fig. 17) of business rules translation into DDL statements, we used the following business rules, considered before in Fig. 9:

- "Each student has full name, student card id, birth date, enrollment date".
- "Each student is given by many scores".
- "Each course has title, semester, approval date".
- "Each course is evaluated by many scores. Each score has value, completion date".

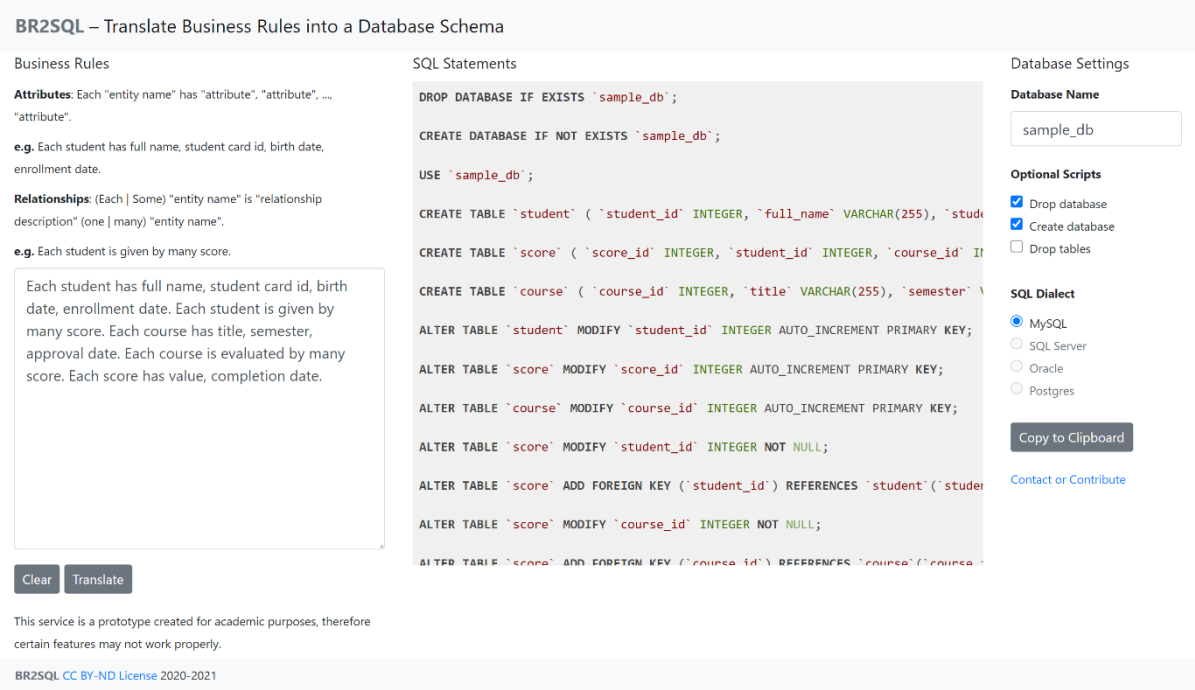

Figure 17: Homepage of the prototype

The DDL scripts obtained for given business rules (see Fig. 17) are basically the same statements demonstrated in Fig. 9 – 11 and Fig. 13. Using these statements, we created the database in MySQL DBMS that is of following scheme (see Fig. 18). The database structure is demonstrated using in-built designer of phpMyAdmin software tool for MySQL administration.

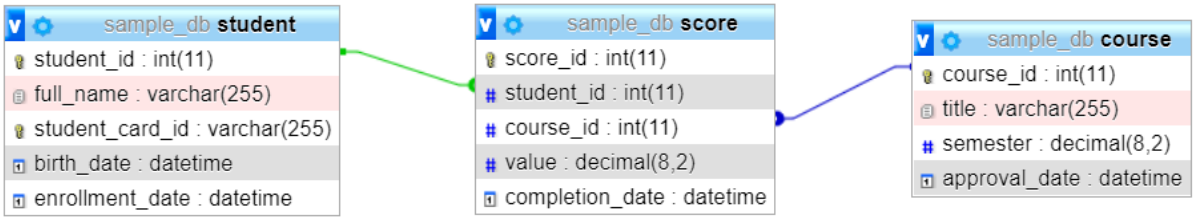

Figure 18: Obtained database structure in phpMyAdmin

After interactive human-computer procedures of column domains and alternate keys selection have been finished, the respective vocabularies were filled with the initial data in order to provide real-time suggestions in future sessions (see Table 1).

Table 1
Initial data of column domains and alternate keys vocabularies

Vocabulary	Classification	Data
Column domains	DateTime	birth_date, enrollment_date, completion_date, approval_date
	Number	value, semester
	Text	full_name, student_card_id, title
Alternate keys	UNIQUE	student_card_id

Another attempt of the same business rules translation will result in the automatically suggested domains, corresponding MySQL data types, and UNIQE alternate keys. Some of classification results by each column domain are outlined below, as well as the alternate key suggestion:

- $\arg\min\{P(DateTime|birth\_date), P(Number|birth\_date), P(Text|birth\_date)\} = \arg\min\{1,0,0\} = DateTime$;

- $\arg\min\{P(DateTime|value), P(Number|Value), P(Text|value)\} = \arg\min\{0,1,0\} = Number$;
- $\arg\min\{P(DateTime|full\_name), P(Number|full\_name), P(Text|full\_name)\} = \arg\min\{0,0,1\} = Text$;
- $student\_card\_id: x = 1, p(x) = 0.73 > 0.5 \Rightarrow$ UNIQUE.

Software prototype keeps training data in JavaScript local storage, which later should be replaced with the database. Business rules executed in further sessions may be translated into DDL statements with automatically recommend column domains and alternate keys. Results of respective suggestion procedures validation are shown in Fig. 19.

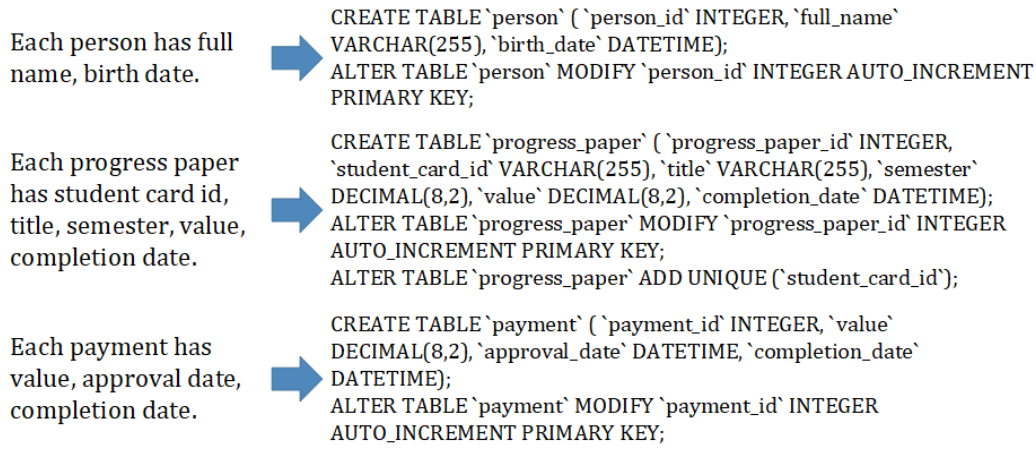

Figure 19: Results of column domains and alternate keys suggestion procedures validation

In order to validate the obtained database structure (see Fig. 18), we have filled tables with sample records (see Fig. 20).

SELECT * FROM `student`

student_id	full_name	student_card_id	birth_date	enrollment_date
1	Patrick G. Harrington	239-09	1988-07-28 00:00:00	2003-07-30 00:00:00
2	Stephen J. Dantzler	389-13	1993-04-30 00:00:00	2011-07-28 00:00:00
3	David B. Bailey	421-94	1998-02-10 00:00:00	2017-07-30 00:00:00

SELECT * FROM `score`

score_id	student_id	course_id	value	completion_date
1	3	1	84.00	2018-01-20 00:00:00
2	3	2	91.00	2018-07-03 00:00:00
3	3	3	78.00	2019-01-12 00:00:00
4	3	4	74.00	2019-06-21 00:00:00
5	3	5	81.00	2020-01-11 00:00:00

SELECT * FROM `course`

course_id	title	semester	approval_date
1	Introduction to Software Engineering	1	2017-05-30 00:00:00
2	Basics of Object-Oriented Programming	2	2017-11-30 00:00:00
3	Introduction to Databases	3	2018-05-30 00:00:00
4	Design of Databases	4	2018-11-30 00:00:00
5	Architecture and Design of Software	5	2019-05-30 00:00:00
6	Design of Information Systems	6	2019-11-30 00:00:00
7	Software Quality	7	2020-05-30 00:00:00
8	Systems Analysis	8	2020-05-30 00:00:00

Figure 20: Sample records stored using the obtained database structure

The following queries and execution results, which show these queries are failed, demonstrate data integrity and consistency validation: created foreign key references ensure strong referential integrity and data consistency according to created unique indexes (see Fig. 21).

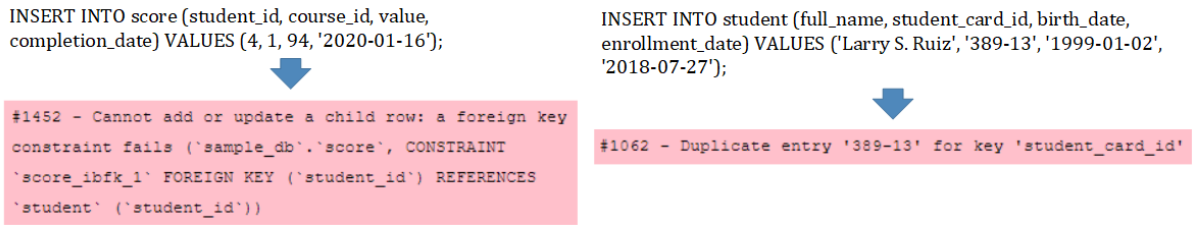

Figure 21: Failed attempts to violate data integrity and consistency

Verification of generated SQL statements was performed automatically when they were executed on MySQL server. However, the database structure could be evaluated using Data Model Scorecard metrics [26]. The most interesting metrics in this case, by our opinion, are 2, 4, and 10 [26] (Table 2).

Table 2

Database structure (Fig. 18) evaluation using some of the Data Model Scorecard metrics

Metric	Total score	Model (Fig. 18) score	Value
How complete is the model?	15	3	0.20
How structurally sound is the model?	15	10	0.67
How well does the metadata match the data?	10	8	0.80

Generated database structure (Fig. 18) is not even close to real-world enterprise databases in the same subject area. The database structure has normalization issue (non-atomic attribute “full_name”). Chosen data types seem proper enough, while sizes of VARCHAR and DECIMAL columns could be specified more precisely. Thus, the obtained database schema (see Fig. 18) was evaluated with metric values demonstrated in Table 2. This particular database was created only for demonstration purposes, however, real-world data models should be refined after evaluation using the Data Model Scorecard metrics [26] or another ones.

5. Conclusion and Future Work

In this paper we have presented the approach and software tool prototype for translation of natural language business rules into database structure. The approach is based on textual descriptions written in a special way, so they could be processed and relational model that contains entities, attributes, and relationships could be obtained. This could be considered as the limitation of the proposed approach, since business rules should correspond to strict patterns even though they are English sentences. Right here the second limitation appears, since proposed approach and software “understand” only English, while multi-language support should be considered in the future research together with more flexible business rules format, which may support at least many-to-many relationships declaration (for now it is required to declare each one-to-many part of the many-to-many relationship separately).

Obtained relational mapping then translated into DDL scripts used to create database schema in a relational DBMS. Data types and unique attributes, referred as alternate keys, since primary keys are generated automatically and supposed to contain auto-increment numbers, are suggested using naïve Bayes classifier and logistic model respectively. This approach requires using vocabularies of titles make assumptions whether certain attribute actually belongs to considered domain or certain attribute indeed a unique key. Future work may include usage of more advanced machine learning and natural language processing methods in order to suggest attribute domains and unique keys, since occurrence of linguistic phenomena, such as plurality, synonymy or polysemy of used attribute titles may lead to suggestion of improper domains and alternate keys. Also correspondence between suggested domains and DBMS data types should be clarified, e.g. using expert judgment methods.

Nevertheless, proposed approach and software tool already demonstrate some initial prototype that may contribute to industry, by simplifying the process of database schema creation and making it less dependent on human errors. Outlined results demonstrate working prototype and its usage to translate sample set of business rules into MySQL database schema. However, research in this field should be

continued to overcome depicted limitations. Also in future work it is planned to consider support of business logic requirements (e.g., constraints, action enablers, inferences, and computations that will be translated into respective database objects, such as triggers, functions, procedures, and views) and referential integrity constraints (i.e., on delete and on update behavior for related tables).

6. References

[1] C. Coronel, S. Morris, Database Systems: Design, Implementation, & Management, Cengage Learning, 2018.
[2] G. K. Gupta, Database Management Systems, McGraw-Hill Education, 2018.
[3] A. Ahmed, B. Prasad, Foundations of Software Engineering, CRC Press, 2016.
[4] O. Pivert, NoSQL Data Models: Trends and Challenges, John Wiley & Sons, 2018.
[5] DB-Engines Ranking, February 2021. URL: https://db-engines.com/en/ranking.
[6] T. Bressoud, D. White, Introduction to Data Systems: Building from Python, Springer Nature, 2020.
[7] W. C. Uduwela, G. Wijayarathna, An Approach To Automate The Relational Database Design Process, Int. J. Database Manag. Syst. 6(7) (2015) 49–56. doi:10.5121/ijdms.2015.7604.
[8] M. Amin et al., Teaching relational database normalization in an innovative way, Journal of Computing Sciences in Colleges 2(35) (2019) 48–56.
[9] D. A. Carpenter, Teaching Tip: Clarifying Normalization, Journal of Information Systems Education 4(19) (2008) 379–382.
[10] T. J. Wang, H. Du, C. M. Lehmann, Accounting for the benefits of database normalization, American Journal of Business Education 1(3) (2010) 41–52. doi:10.19030/ajbe.v3i1.371.
[11] B. Mathew, Performance Evaluation of 3rd Normal Form Decompositions : master thesis, Florida Institute of Technology, 2019.
[12] K. E. Wiegers, J. Beatty, Software Requirements. Best practices. Developer Best Practices, Microsoft Press, 2013.
[13] The Economics of Testing. URL: http://www.riceconsulting.com/public_pdf/STBC-WM.pdf.
[14] M. Stoica, M. Mircea, B. Ghilic-Micu, Software development: Agile vs. traditional, Informatica Economica, 4(17) (2013) 64–76. doi:10.12948/issn14531305/17.4.2013.06.
[15] A. Rayskiy, Why should testing start early in software project development?, 2017. URL: https://xbsoftware.com/blog/why-should-testing-start-early-software-project-development/.
[16] I. S. Bajwa, M. G. Lee, B. Bordbar, SBVR business rules generation from natural language specification, 2011 AAAI Spring Symposium Series, 2011, pp. 2–8.
[17] S. Moschoyiannis, A. Marinos, P. Krause, Generating SQL queries from SBVR rules, International Workshop on Rules and Rule Markup Languages for the Semantic Web, Springer, Berlin, Heidelberg, 2010, pp. 128–143. doi:10.1007/978-3-642-16289-3_12.
[18] A. Kate et al., Conversion of Natural Language Query to SQL Query, 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), IEEE, 2018, pp. 488–491. doi:10.1109/ICECA.2018.8474639.
[19] A. M. Kopp, D. L. Orlovskyi, Towards the approach to database structure generation from business rules based on natural language expressions, Information technologies and automation 2020, October 22-23, 2020, pp. 224-226. doi:10.5281/zenodo.4172923.
[20] A. Meduna, Automata and Languages: Theory and Applications, Springer Science & Business Media, 2012.
[21] Schema.org. URL: https://schema.org/.
[22] S. Raschka, Naive Bayes and Text Classification I – Introduction and Theory, 2014. URL: https://arxiv.org/pdf/1410.5329.pdf.
[23] C. E. Nwankpa et al., Activation Functions: Comparison of Trends in Practice and Research for Deep Learning, 2018. URL: https://arxiv.org/pdf/1811.03378.pdf
[24] business_rules2sql GitHub repository. URL: https://github.com/andriikopp/business_rules2sql.
[25] BR2SQL – Translate Business Rules into SQL. URL: https://br2sql.herokuapp.com/.
[26] D. Henderson, S. Earley, DAMA-DMBOK: Data Management Body of Knowledge, Technics Publications, 2017.