Generating SQL-Query-Items Using Knowledge Graphs

Paul L. Christ[1][a], Torsten Munkelt[2] and Jörg M. Haake[1][b]
[1]*Faculty of Computer Science and Mathematics, Distance University Hagen, Universitätsstraße 11, Hagen, Germany*
[2]*Facult ermany*
{*firstname.lastname*}*@fernuni-hagen.de,* {*firstname.lastname*}*@htw-dresden.de*

Keywords: SQL, Relational Database, AIG, Automatic Item Generation, Knowledge Graph, Large Language Model.

Abstract: SQL is still one of the most popular languages used in todays industry across many fields. Poorly written SQL remains one of the root causes of performance issues. Thus, achieving a high level of mastery for SQL is important. Achieving mastery requires practicing with many SQL assessment items of varying complexity and content. The manual creation of such items is very labor-some and expensive. Automatic item generation reduces the cost of item creation. This paper proposes an approach for automatically generating SQL-query items of varying complexity, content, and human-like natural language problem statements (NLPS). The approach is evaluated by human raters regarding the complexity and plausibility of the generated SQL-queries and the preference between two alternative NLPS. The results show agreement on the plausibility of the generated SQL-queries, while the complexity and the NLPS preference show higher variance.

1 INTRODUCTION

Despite recent advances in natural language interfaces to relational databases (Li et al., 2024), SQL ranks among the most popular languages used in industry (Stephen Cass, 2024). The Association for Computing Machinery (ACM) recommends the learning of SQL in its current computing curricula recommendations, and SQL is also relevant in other disciplines, such as mechanical engineering (Cc2020 Task Force, 2020). Since poorly written SQL queries are often one of the root causes of performance issues in applications, attaining a high level of mastery for SQL is important for many professional roles. Achieving a high level of mastery for SQL requires deliberate practice (Hauser et al., 2020). Deliberate practice is facilitated by solving many tasks with well-defined learning objectives, personalized feedback and a task difficulty that exceeds the competency-level of the learner slightly (Ericsson, 2006). Other educational theories, such as scaffolding and the zone of proximal development (ZPD), support the underlying concepts of the theory of deliberate practice further (Raslan,). To enable students to gain deliberate practice for mastering SQL, a large amount of SQL assessment items is required so that students have a chance to repeat and choose items of appropriate difficulty and content. The manual creation of such items is labor-some and expensive (Westacott et al., 2023). Automated item generation (AIG) can significantly reduce the cost of creating large amounts of items, without losing item quality (Kosh et al., 2018).

This article presents an approach for generating SQL-query items using knowledge graphs. Each item consists of a generated SQL-query, a generated natural language task description, and a depiction of the underlying database schema. A user study evaluates the approach by assessing aspects of the generated SQL-query and the task description in natural language.

The remainder of the paper is organized as follows. Section 2 illustrates the structure of SQL-query items and specifies the requirements for the generation of such items. Section 3 provides an overview of the current state of the art of, firstly, the generation of SQL-queries and, secondly, the generation of the descriptions of SQL-queries in natural language, and compares the state-of-the-art with the above requirements, and lastly section 4 presents the approach for generating SQL-query items using Knowledge Graphs. Section 5 presents the evaluation setup, results and their discussion. Section 6 concludes the paper and gives an outlook on future work.

[a] https://orcid.org/0000-0002-6096-7403
[b] https://orcid.org/0000-0001-9720-3100

Christ, P. L., Munkelt, T. and Haake, J. M.
Generating SQL-Query-Items Using Knowledge Graphs.
DOI: 10.5220/0013498300003932

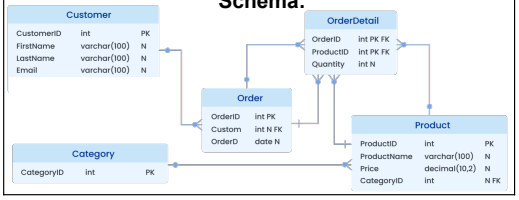
In *Proceedings of the 17th International Conference on Computer Supported Education (CSEDU 2025) - Volume 1*, pages 785-796
ISBN: 978-989-758-746-7; ISSN: 2184-5026

Input:

Task Description:
Write a SQL-query that extracts the described data from the database schema shown below.

Problem Statement:
Find the e-mail of all customers that ordered swimming related products between 90 and 100 dollars.

Schema:



Expected Output:

Reference-Query:

```
SELECT Email
FROM Customer AS c
JOIN ORDER o ON c.CustomerId = Custom
JOIN OrderDetail od ON o.OrderId = od.OrderId
JOIN Product p ON od.ProductId = p.ProductId
JOIN Category ca ON p.CategoryId = ca.CategoryId
WHERE ca.CategoryId = 2 AND
      p.Price BETWEEN 90 AND 100;
```

Expected Result Set:

Email
Gregory.D@gmail.com
I.Brown@yahoo.com
Se.Ku@web.de
Flo.S@gmx.net

Figure 1: Structure of an examplatory SQL-query item.

2 REQUIREMENT ANALYSIS

2.1 Example SQL-Query-Item

Figure 1 illustrates the structure of a SQL-query-item. Each item comprises an input that contains the information required to understand and solve the task, and an expected output. The input consists of a static task description that explains the task goal, a natural language problem statement (NLPS) that formulates what data to extract from the database, and the database schema as a reference. The expected output consists of a reference query and a corresponding expected result set to be compared with the answer of the learner.

2.2 Requirements

Requirements for the attributes of SQL-query items and the generation process are derived from example SQL-query items and the experiences of the researchers.

R0. The algorithm shall generate complete items, as shown in figure 1, based on a database schema with annotations and generation parameters provided by a teacher. A complete item includes the predefined task goal, an NLPS, the database schema, a reference query and the expected result set.

R1. The algorithm for the generation of SQL-queries shall offer parameters to allow the teacher to regulate the complexity of the generated SQL-query.

R2. The algorithm for the generation of SQL-queries shall offer parameters to allow the teacher to influence the diversity of the content of generated SQL-queries so that items adress specific learning objectives.

R3. The algorithm for the generation shall only generate SQL-queries that adhere to standard SQL syntax in general and specifically the Data Query Language subset (ISO Central Secretary, 2016).

R4. The algorithm shall generate SQL-queries that are semantically plausible. *R4.1* The generated SQL-query shall be coherent, i.e. the query doesn't contain redundant constituents, the combination of constituents doesn't contradict the behavior of one or more other constituents, **and** every constituent effects the result set. *R4.2* The generated SQL-query shall be acceptable, i.e. the query is deemed plausible **and** is likely to be encountered in a real-world scenario.

R5. For each generated SQL-query item, the system shall generate an NLPS that enables the learner to create a query that produces the expected result set. Therefore the NLPS should be unambiguous and contain all relevant information.

R6. The generated problem statements shall be formulated in *human-like* natural language.

What makes machine-generated text *human-like* differs depending on domain and task, and the task-dependent evaluation often lacks concrete specifications (van der Lee et al., 2021). Thus, for the task of generating NLPS that match SQL-queries, two criteria are defined to determine the property of *human-likeness*: *R6.1* The clarity of the NLPS is considered high, if it exhibits a low amount of missing information, a high quality of writing style and high comprehensibility. *R6.2* The conciseness of the NLPS is considered high, if it exhibits a short text length, low information redundancy and completeness of information required for solving the item.

The conciseness-criteria aim to reflect the general human trait to omit information that seems self-evident, also referred to as *tacit knowledge* (Becker et al., 2021), as well as the manner in which a set of data would be described in a practical setting, which is rather the question of *what* data is required instead of *how* the data should be extracted.

R7 The formulation must not contain any ambiguities. Ambiguities in the NLPS would make it unfeasible to, in general, infer an SQL-query that generates the same output as the generated SQL-query according to the underlying database (schema).

R8 The algorithm for generating SQL-query items shall require minimal human effort for *R8.1* creating the generation input of the SQL-query generation and for *R8.2* generating the NLPS.

R9 The code for generating SQL-query items and generating NLPS from SQL-queries shall be openly available for verifiability and reuse.

3 RELATED WORK

3.1 Generation of SQL-Queries

Only related work on AIG for SQL-query items in educational settings is considered. General SQL-query teaching tools and frameworks, which do not generate the exercises, are also not considered, as we focus on the generation of SQL-query-items instead of considering SQL-specific teaching techniques.

Approaches for the generation of SQL-queries can be further subdivided by the method they utilized. We differentiate between rule-based approaches and approaches based on generative artificial intelligence.

Several rule-based approaches exist for generating SQL-query assessment items. (Gudivada et al., 2017) propose a theoretical adaptation of context-free grammar for SQL, but the generated queries lack plausibility, semantic depth, and parameterization. (Do et al., 2014) develop a metadata-driven method allowing for broad SQL-constituent selection and multi-table JOINs, yet their queries also suffer from low plausibility and semantic depth. (Atchariyachanvanich et al., 2019) introduce the RSQLG algorithm, which generates SQL queries and corresponding NLPS using predefined templates. While similar to (Do et al., 2014), it operates on a more limited SQL subset and single-table queries, leading to issues with plausibility and typicality. (Chaudhari et al., 2021) extend this approach to include DDL and DML commands.

(Aerts et al., 2024) experiment with GPT-3.5 to generate SQL-query items. The approach utilizes different prompts to elicit SQL-queries varying in content. The results were evaluated regarding multiple aspects, such as *Sensibleness*, *Novelty*, *Topicality* and *Readiness*. Despite the small sample size of 5 examples per prompt, errors for the generated SQL-queries occur in all aspects and require manual correction.

Table 1 states if the discussed approaches fulfill the requirements discussed in section 2. The mapping of the numbers to the approaches is as follows: (Gudivada et al., 2017), (Do et al., 2014), (Atchariyachanvanich et al., 2019), (Chaudhari et al., 2021), (Aerts et al., 2024). The symbols $+$, $\sim$ and $-$ indicate that the requirement is fulfilled, partially fulfilled or not fulfilled. Only one approach fulfills requirement *R0* fully. Most approaches fulfill requirement *R1*. Only one approach fulfills requirement *R2* fully. Almost all approaches fulfill *R3* fully. Only one approach fulfills *R4.1* partially, all other approaches fail to fulfill *R4.1* and *R4.2* Most approaches fulfill *R5*, but fail to fulfill *R6.1*, *R6.2* and *R7* fully. Some approaches fulfill *R8.1* and *R8.2*. Only one approach fulfills *R9* partially.

Table 1: Comparison of Approaches for the Generation of SQL-Queries Regarding the Corresponding Requirements.

	R0	R1	R2	R3	R4.1	R4.2	R5	R6.1	R6.2	R7	R8.1	R8.2	R9
[1]	−	+	−	+	−	−	−	−	−	−	−	−	−
[2]	−	+	−	+	−	−	−	−	−	−	−	−	−
[3]	+	+	∼	+	−	−	+	∼	∼	+	+	+	−
[4]	∼	−	∼	+	−	−	+	∼	∼	∼	+	+	−
[5]	∼	∼	+	−	∼	−	+	−	+	∼	∼	∼	∼

3.2 Generating Natural Language Problem Statements from SQL-Queries

We categorize the approaches for SQL-to-text into template- and rule-based and neural machine translation. Rule-based approaches explicitly employ the information represented by the query, along with predefined text-templates, to create a corresponding NLPS. Neural machine translation is usually used for translating a natural language into another natural language, but can also be utilized to transform a structured language into natural language or vice versa. This typically involves a large amount of training data, but doesn't require the manual construction of transformation rules.

Several approaches generate NLPS from SQL queries with rule- or template-based methods. (Koutrika et al., 2010) transform SQL queries into directed graphs and apply predefined text templates, requiring significant manual effort to model database schemas. Their method ensures correctness and extensibility but demands high setup effort. (Kokkalis et al., 2012) extend this by enabling multilingual descriptions and introducing a GUI for schema annotation. (Eleftherakis et al., 2021) further refine the approach by improving template-based fluency and supporting additional SQL constituents.

Neural machine translation approaches have been explored for generating natural language descriptions from SQL queries. (Iyer et al., 2016) use a sequence-to-sequence model with LSTM encoding and feed-forward decoding but achieve only $\sim 63\%$ accuracy on simple single-table queries. (Xu et al., 2018) im-

prove accuracy to $\sim$ 75% by employing a graph-to-sequence model with an attention-based decoder for DAG-structured SQL queries. (Ma et al., 2021) extend this approach using a transformer-based model with custom attention strategies, enabling support for complex queries with subqueries and multiple JOINs, though accuracy drops to $\sim$ 66%.

Table 2: Comparison of Approaches for the Generation of Natural Language Problem Statements from SQL-Queries.

	R5	R6.1	R6.2	R7	R8.2	R9
(Koutrika et al., 2010)	$\sim$	+	+	+	−	−
(Kokkalis et al., 2012)	$\sim$	+	+	+	−	−
(Eleftherakis et al., 2021)	+	+	+	+	$\sim$	−
(Iyer et al., 2016)	$\sim$	−	+	−	$\sim$	+
(Xu et al., 2018)	$\sim$	$\sim$	+	−	$\sim$	−
(Ma et al., 2021)	+	$\sim$	+	−	$\sim$	+

Table 2 compares the discussed approaches for the generation of NLPS regarding a subset of the corresponding requirements discussed in section 2. The subset consists of requirements *R5* to *R7*, *R8.2* and *R9* as only they apply to the generation of NLPS. Again, the symbols +, $\sim$ and − indicate that the requirement is fulfilled, partially fulfilled or not fulfilled. Only two approaches fulfill requirement *R5* fully. Only the rule-based approaches fulfill the requirements *R6.1* and *R7*. All approaches fulfill requirement *R6.2*. *R8.2* is fulfilled only partially by some approaches. *R9* is only fulfilled by two of the neural machine translation approaches. The approach introduced by (Eleftherakis et al., 2021) performs best, as it fulfills requirements *R5* to *R7* fully and *R8.2* partially.

3.3 Deficits of Related Work

As seen in sections 3.1 and 3.2, no approach exists that fulfills all the requirements defined in section 2 fully. The approaches for generating SQL-query items especially fail to fulfill requirement *R4*. While some approaches fulfill requirement *R5*, they do so with relatively poor quality, regarding the aspects defined in requirement *R6*. The approaches for generating NLPS from SQL-queries fulfill the required subset of requirements *R5* to *R7* and *R8.2* to *R9* more often.

The best performing approaches for generating SQL-query items and NLPS don't fulfill *R9* and therefore can not be extended to fulfill the remaining requirements. The ideas and concepts introduced by the best approaches are utilized in our own approach. We extend those ideas and concepts to fulfill the remaining requirements *R2*, *R4.1*, *R4.2*, *R8.2* and *R9*.

4 GENERATION OF SQL-QUERY-ITEMS

4.1 Overview of the Generation System

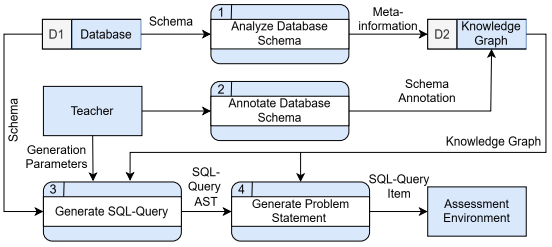


Figure 2: Level 1 DFD of the SQL-query-item generation.

Figure 2 illustrates the proposed system for generating SQL-query items with a data flow diagram (DFD). When a new database is added to the system, the system analyzes its schema to extract meta-information about tables, attributes and foreign-keys and stores it in the Knowledge Graph (KG). A teacher then manually annotates the database schema with semantic labels for entities and their relationships. The annotations are stored at the corresponding elements of the KG. Once a schema has been analyzed and annotated, a teacher can trigger the SQL-item generation, which consists of the processes 3 and 4 and receives a set of generation parameters, the database schema and the KG. First, a SQL-query is generated on the basis of the generation parameters and the meta-information about the schema in the KG. Then, the corresponding NLPS is generated based on the abstract syntax tree (AST) of the SQL-query and the annotated schema in the KG. In the end, the whole SQL-query-item is handed to the teacher for the use in the assessment environment.

4.2 Knowledge Graph Construction

4.2.1 Database-Schema-Analysis

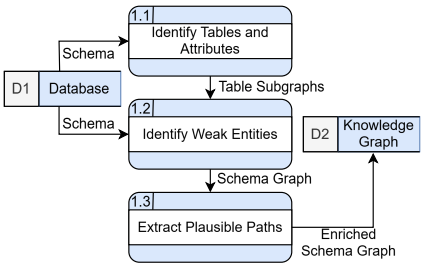


Figure 3: Level 2 DFD of the database schema analysis.

Figure 3 decomposes process 1 *Analyze Database Schema* into the subprocesses 1.1 *Identify Tables and Attributes*, 1.2 *Identify Weak Entities* and 1.3 *Extract*

Plausible Paths. The subprocesses result in the construction of a graph representation of the database schema that encodes meta-information about tables, attributes and the foreign-key relation between tables.

Subprocess 1.1 receives a schema from the database and identifies the corresponding set of all tables and their corresponding attributes. A directed graph is initialized for the schema. The graph receives the tables and the attributes as nodes. The datatype of each attribute and whether it is part of a primary or a foreign key is stored as labeled properties at the attribute's node. Between attribute nodes and their table nodes *part-of* relationships are added as directed edges to the graph. The type of the respective relationship is stored as a labeled property at the edge. The result is a graph of disconnected subgraphs, one for each table and its corresponding attributes.

Subprocess 1.2 receives the graph of identified table subgraphs from subprocess 1.1 and the corresponding schema from the database. Then, the foreign-key constraints of each table are inspected, to identify weak entities. A weak entity is an entity that cannot be uniquely identified by its attributes alone (Elmasri and Navathe, 2016). A table therefore represents a weak entity if its primary key contains at least one foreign-key. Weak entities can be further divided into associative and subtype entities. A table represents an associative entity if its primary key contains more than one foreign key. A table represents a subtype entity if it is a weak entity and contains a discriminatory attribute that is not part of the primary key. An associative entity encodes a *many-to-many* relationship between entities, of which an attribute is referenced as a foreign key by and used as part of the associative entities primary key by breaking them into multiple *one-to-many* relationships. This means that three edges exist for an associative entity that connects two primary entities. One between each primary entity and the associative entity, and one edge which encodes the conceptual (many-to-many) relationship between the two primary entities. A subtype entity forms an *is-a* relationship between the subtype entity and the primary entity. The mentioned relationships are added as edges to the graph. The type of relationship is stored as a labeled property of the edge. The result is a graph of the connected subgraphs, where the subgraphs represent tables and their corresponding attributes, and the edges connecting the subgraphs represent the foreign key constraints (or resulting conceptual relations) between the tables.

Subprocess 1.3 receives the schema graph from subprocess 1.2 and extracts semantically plausible (JOIN-)paths in the schema graph. As described in section 2, we consider a query semantically plausible if it is coherent, meaningful and typical. We further assume that the set of all possible queries for a schema underlies a left-tailed distribution, where the frequency with which a query is asked (typicality) lies on the x-axis and the meaningfulness of the query lies on the y-axis (Mandamadiotis et al., 2024). To increase the plausibility of generated queries, the generation should not generate most of the queries in the long tail of that distribution. To reduce the problem space in a way that removes mostly implausible queries (cut the tail), we only consider paths in the schema graph as plausible iff they contain at least one primary entity and do not end in an associative entity. This way, we increase the meaningfulness of the queries by reducing the effects of database normalization that splits conceptual entities into separate tables. The (JOIN-)paths are determined by graph traversal using the identified primary entities as starting points. The plausible paths are then marked in the schema graph. The entire enriched schema graph is then stored in the KG.

4.2.2 Manual Schema Annotation

Each table, attribute and relationships between tables have a conceptual meaning that is not captured in the limited semantics of relational databases. In order to produce human-like NLPS as described in section 2, conceptual labels are required. To achieve reliable labels, a manual annotation process is employed. The annotations include conceptual labels for attributes, tables and the relationships between tables. In the case of a table being an associative entity, only the conceptual path between the entities is annotated. The schema annotations provided by the teacher are then stored as label properties for the respective elements of the knowledge graph.

4.3 SQL-Query Generation

4.3.1 Parameters for Generating SQL-Queries

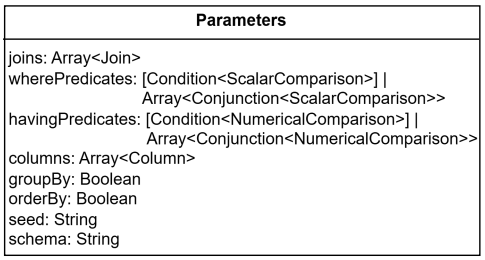


Figure 4: Top level of the input parameter data type.

Due to the vast space of possibilities the *SELECT*-command offers, the available SQL-constituents for

the generation are reduced to a smaller subset [1]. Subqueries are not included. The top level data type of the input parameters is shown in the data structure diagram in figure 4.

The parameters cover the configuration of the SQL-constituents *joins*, *where-* and *having-predicates*, selected *columns*, *group-by* and *order-by*, a *seed* for making the generation deterministic, and the selection of a database *schema* [2].

The parameters allow for a fine-grained specification of the supported SQL-constituents in the generated SQL-query. The number of joins, where- and having-predicates in the generated query is governed by the number of elements in the array of the respective parameter.

For example, if the parameter *joinSelection* has the value of an empty array $[]$, no joins are included in the generated query. If the parameter *joinSelection* has the value of $[\{\}, \{\}]$, two joins are included in the generated query, but the specific characteristics of the joins are chosen randomly.

The parameters can be easily extended to support additional SQL-constituents for the generation.

The satisfiability of a certain parameter configuration depends on the utilized schema and the contained data. E.g. if two joins over three tables are required and the database schema contains only two tables, the parameter is not satisfiable. If the schema contains three or more tables, the parameter is satisfiable. If all parameters of a configuration are satisfiable, the configuration is satisfiable.

To handle unsatisfiable parameters, the parameters are considered to be soft constraints whose satisfiability is decided during the generation process and may be violated. Unsatisfiable parameters are handled gracefully by relaxing the constraint. The type of an unsatisfiable parameter P_i is inferred via runtime reflection, and the remaining potentially satisfiable parameter options are determined. The first satisfiable option is returned. If no option is satisfiable, the unsatisfiable parameter is omitted from the configuration.

4.3.2 Constraint-Adhering Join-Path-Selection and Clause-Generation

The decomposed SQL-query generation process is shown in figure 5. Subprocess 3.1 receives the generation input, consisting of the generation parameters, the database schema governed by the *schema* parameter and the knowledge graph (KG). For simplification of the process, the generation input is stored in a separate data sink and accessed by subsequent processes. Subprocess 3.1 then triggers the actual SQL-query generation process by invoking subprocess 3.2 with the generation input.

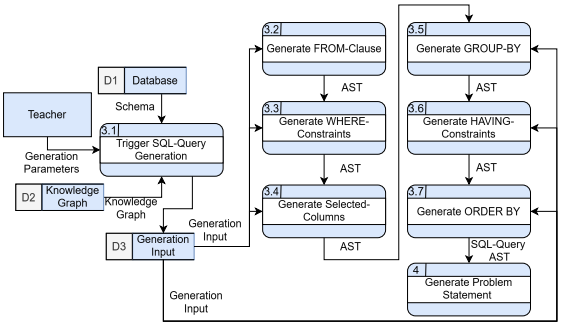


Figure 5: Level 2 DFD of the SQL-query generation.

Subprocess 3.2 receives the generation input and generates the *FROM*-clause of the SQL-query. The generation is governed by the *joinSelection* parameter. If it contains no elements, a random table of the schema is selected for the inclusion in the FROM-clause. If the parameter contains one or more elements, the KG is searched for paths that match the requested join length, and one such path is selected at random. If no paths of that length exists, the path length is reduced until a path is found or only a single table is selected. If the elements of the *joinSelection* parameter contain further specifications regarding the join-type, it is included in the generation of the current join statement of the *FROM*-clause. If no specifications are given, a random join-type is chosen. The current system only supports equi-joins and accepts only tables other than itself as a join-target. The system can be expanded to allow other join-targets, such as subqueries and self-joins, and more operators for increasing the variety of join-conditions. The process results in an abstract syntax tree (AST) representation of a *FROM*-clause.

Subprocess 3.3 receives the AST from subprocess 3.2 and the generation input. The generation is governed by the *whereP redicateSelection* parameter. If it contains no elements, no WHERE-clause is generated. If the parameter contains one *Condition*-element a singular condition is generated, either with a given relational operator or, if absent, with a random operator. The condition target is chosen randomly from the attributes of the tables in the *FROM*-clause and their available values. For comparison operations, one or more values are sampled from all values that are present in the database for that attribute. If the parameter contains two or more *Conjunction*-elements,

[1] Available at https://github.com/HTW-ALADIN/SQL-Query-Generation

[2] The concrete data types are defined at https://github.com/HTW-ALADIN/SQL-Query-Generation

multiple conditions are generated as described before. The conditions are combined either by specified or random conjunction-types (*AND* or *OR*). The result of subprocess 3.3 is an AST extended with a possibly empty WHERE-clause.

Subprocess 3.4 receives the AST from subprocess 3.3 and the generation input. The generation is governed by the *columnSelection* parameter. If it contains no elements, all attributes of the tables in the *FROM*-clause are chosen ("*"). If the parameter contains one or more *Columns*, the columns are selected according to the specification or, if absent or unsatisfiable, chosen at random from the available attributes of the tables from the *FROM*-clause. Columns may be specified to apply an aggregation operator to them. If no aggregation operator is specified, no random aggregation operator is selected. If an aggregation operator but no column-type is specified, we impose certain restrictions to increase the plausibility of the query, such as not allowing aggregates on ID-columns, as the average of an automatically incremented ID does not make sense. The result of the subprocess 3.4 is an AST extended with the *SELECT*-statement.

Subprocess 3.5 receives the AST from subprocess 3.4 and the generation input. The generation is governed by the *groupBy* parameter. If it is set to true, a *GROUP-BY* clause is generated. If no aggregation operations are generated in subprocess 3.4, a *group-by* clause is generated that includes all selected columns. This has essentially the effect of the *DISTINCT* keyword. We allow this behavior because we consider it semantically plausible and do not implement the *DISTINCT* keyword in the generation process. If an aggregation operator was generated, the generated *GROUP-BY* clause contains all selected columns, that are not aggregated. The result of the process is an AST extended with the *GROUP-BY*-clause.

Subprocess 3.6 receives the AST from subprocess 3.5 and the generation input. The generation is governed by the *havingPredicateSelection* parameter. If it contains no elements, no *HAVING*-clause is generated. If it contains a single *Condition*-element, a singular condition is generated, either according to the specification or, if absent, at random. The condition target is chosen randomly from the attributes of the tables in the *FROM*-clause. The generation always applies a random aggregation operator to the target, as the *HAVING*-clause operates on the table expression as a set. Hence we do not consider a *HAVING*-clause applied to non-aggregated columns semantically plausible, even if the standard allows it. If no *group-by*-clause was generated by subprocess 3.5 and multiple columns were selected by subprocess 3.4, no *HAVING*-clause is generated, as it would not be valid SQL. If the *havingPredicateSelection*parameter contains one or more *Conjunction*-elements, multiple conditions are generated as described before. The conditions are combined either by specified or random conjunction-types (*AND* or *OR*). The result of subprocess 3.6 is an AST extended with the *HAVING*-clause, if it was generated.

Subprocess 3.7 receives the AST from subprocess 3.6 and the generation input. The generation is governed by the *orderBy* parameter. If it is set to true, an *ORDER-BY*-clause is generated, with a randomly chosen order (*ASCENDING* or *DESCENDING*). Subprocess 3.7 results in an AST extended with the *ORDER-BY*-clause, if it was generated, and ends the process of the SQL-query generation.

4.4 Generating NLPS

Our approach to generate NLPS follows similar concepts as proposed in (Eleftherakis et al., 2021). We also utilize a template-based approach with additional (optional) semantic labels for the entities, their attributes and relationships, as described in section 4.2.2. We also treat the SQL-query as a graph, albeit as a tree structure, and traverse it to compose a text from the text-template associated to the current tree element and the tree element itself. We extend the approach by using a large language model (LLM) as a surface realization engine, as proposed in (Farahnak et al., 2020). We utilize two sets of predefined templates[3]. One template set is used by the baseline approach, that does not use the meta-information and semantic labels stored in the knowledge graph (KG). The second template set is used by a hybrid approach, that uses the meta-information and semantic labels stored in the KG.

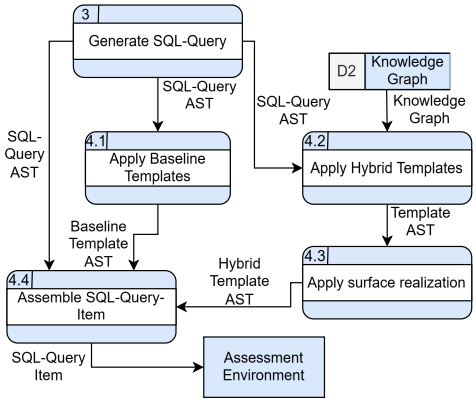


Figure 6: Level 2 DFD of generating the NLPS.

[3]Both template sets are available at https://github.com/HTW-ALADIN/SQL-Query-Generation

The decomposed process for generating NLPS is shown in figure 6. Subprocess 4.1 receives the AST representation of the SQL-query. The AST is then traversed and for each subtree or node a baseline-template is selected. For each template, the required entities are extracted from the corresponding subtree or node of the AST. The applied templates are directly handed over to subprocess 4.4.

Subprocess 4.2 receives the AST representation of the SQL-query and the KG. The AST is traversed and for each subtree or node a hybrid-template is selected. For each template, the required entities are replaced by their corresponding semantic label. In the case of two inner equi-joins that connect two primary entities via an associative entity, the semantic label of the relationship replaces both templates describing the joins. The templates are then handed over to subprocess 4.3.

Subprocess 4.3 receives the AST representation of the applied hybrid templates. The hybrid template may contain multiple *[MASK]*-tokens. These tokens are placed in the template instead of certain part-of-speech-elements, such as determiners or pronouns. The AST is traversed again, and each *[MASK]*-token is replaced by a LLM. We utilize the LLM BART for this task (Lewis et al., 2019). The templates are then handed over to subprocess 4.4.

Subprocess 4.4 receives the AST representation of the SQL-query, the applied baseline-templates and the applied hybrid templates. The process traverses each AST again and joins the individual templates together to a single string. In the case of the SQL-query AST, a standard template set is used for the PostgresDB SQL dialect. All artifacts of the SQL-query item, the task description that explains the task goal, the NLPS, the database schema as a reference, and the expected output (reference query and a corresponding expected result set) are then bundled and handed over to the assessment environemnt, which may display a task description, NLPS and database schema to the learner, to offer the learner a way to input a solution, and display the result of the comparison between expected result and learner solution to the learner.

5 RESULTS AND DISCUSSION

5.1 Evaluation Setup

Functional testing of the implementation of the proposed solution showed that the implementation fulfills the functional requirements. Thus, this section focuses on validating the fulfillment of the non-functional requirements that were unfulfilled by the related work specified in section 3.3. For this purpose, we conducted a survey to evaluate the fulfillment of R4.2. Additionally, we were interested in exploring the preferences of teachers regarding two variants (baseline and hybrid) of NLPS, which were generated by the approach shown in section 4.4. Lastly, we examined the complexity of the generated queries in the study, to verify *R1* through human judgment.

5.1.1 Survey Structure

The survey items are structured as follows: each survey item shows the database schema, the generated SQL-query, the corresponding result set and the two NLPS. For each survey item, the following questions were asked: 1.) How plausible is it for a human to request the information expressed by the shown SQL-query for this database? 2.) How frequently would you assume that the information expressed by the shown SQL-query to be requested for this database? 3.) How complex would you rate this query? 4.) Which text reads more natural (1 or 2)?

Questions 1. and 2. were rated with a five point Likert scale. Question 3. was rated on a scale between 1 and 10. Question 4. was rated with a binary choice.

Consistent high scores for question 1. and 2. would indicate the fulfillment of *R4.2*. Question 3. explores the complexity of the generated SQL-query items. *R2* would seem fulfilled if the complexity of items are rated consistently and the complexity between items cover the whole spectrumn. Question 4. explores the stylistic preference of raters regarding the two NLPS.

Each item is rated by three raters. To dissolve potentially inconclusive ratings, we employ three strategies. The first strategy is to calculate the mean of all ratings per question. This strategy is utilized for the analysis of question 3.). The second strategy is to determine the majority vote in the case of two raters providing the same rating. The third strategy is to determine the mean of all ratings per question and round it to the nearest integer. Strategy two is utilized for question 1.) and 2.) if two raters provide the same rating, and for question 4.). Strategy three is utilized for question 1.) and 2.) if all ratings are different. We acknowledge the bias the third strategy introduces to ordinal scaled data, as in question 1.) and 2.).

5.1.2 Study Procedure

We utilized Amazon MTurk as an evaluation platform. Anonymous raters were selected via a pretest. The pretest consisted of five single choice questions that were manually created by the researchers that conducted the study. Each pretest item consists of a general task description, a depiction of a database

schema, an NLPS, an expected result set that matches the NLPS and four answer options of which three are distractors. Each answer option is a SQL-query. The distractors were created by removing parts of the correct query or adding additional keywords and clauses. The task description instructs the rater to select the answer option that produces the shown expected result set for the given NLPS and the provided schema. There were no preconditions to be able to complete the pretest. Only raters that successfully answered all five single choice questions were allowed to rate the survey items. Each survey item was evaluated by three raters. A total amount of 639 different raters took part in the survey. Each rater rated on average three items. 90% of the raters rated between 1 and 5 items, and the remaining 10% rated between 6 and 60 items each. Each rater received 0.40$ per completely rated item and for the completion of the entire pretest. After the completion of an item, raters were not explicitly prompted to complete another item. If they wanted to complete another item, they had to manually select it from their MTurk dashboard showing their eligible tasks.

5.1.3 Metrics for Assessing the Textual Quality

To evaluate the two generated variants of NLPS further, additional text quality metrics are utilized. We utilize different readability metrics, such as the text length, the term frequency and the Flesch Reading Ease score (Flesch, 1948). Lesser text length, lesser term frequency and higher Flesch Reading Ease score indicate improved readability. The semantic similarity between the SQL-query and the corresponding NLPS is measured by encoding both into a text embedding and calculating the pairwise cosine similarity of the resulting embedding vectors. We utilized SBERT as a sentence embedding for measuring the semantic similarity (Reimers and Gurevych, 2019). A higher percentage indicates higher semantic similarity. The overall text quality of the NLPS is judged by a suite of heuristic quality metrics and text repetition metrics, and tested against thresholds for the suite of metrics as defined in (Raffel et al., 2023). The text quality is considered sufficient only if the NLPS hits the thresholds for all quality metrics. The syntactic complexity of the NLPS is measured by the dependency distance. The dependency distance calculates the average distance from each lexical unit to their dependent recursively, such as token-to-token up to sentence-to-sentence distance (Oya, 2011). A lower dependency distance indicates lower syntactic complexity. The coherence of the NLPS is measured by the semantic similarity between consecutive sentences. A higher percentage of semantic similarity indicates greater coherence.

5.1.4 Composition of the Evaluation Dataset

The evaluation dataset consists of 667 generated unique SQL-queries and their corresponding generated NLPS [4]. The database schema for which the queries were generated is an adaptation of the IMDB database[5].

For the purpose of the evaluation each query is split into the seven categories: selected columns, used aggregation functions, used tables, where predicates, having predicates, group by and order by. The value of a query for each category is the amount of keywords the query contains that can be assigned to each category. The result of this aggregation yields 140 distinct combinations of keyword amounts for the respective category. 48 of the combinations are unique. 44 of the combinations have a frequency of occurrence of 2 to 3 times. 36 of the combinations have a frequency of occurence of 4 to 10 times. The remainding 12 combinations have a frequency of occurence of 11 to 25 times.

The generation parameters that affect the dimensions columns, tables, where and order were selected randomly. For a subset of 600 queries, the parameters affecting the dimensions aggregates, having and group were kept at zero and then for the remaining subset of 67 queries the parameters were also selected randomly. This split was chosen to focus specifically on the dimensions *Columns*, *Tables* and *Where*.

Table 3: Inter Rater Agreement (IRA) per question.

	Q1	Q2	Q3	Q4
IRA	0.75	0.87	0.86	0.51
Weight	Ordinal	Ordinal	Ordinal	Identity

5.2 Evaluation Results

5.2.1 Inter-Rater Agreement

We utilize weighted Krippendorf's alpha (Krippendorff, 2004) to estimate the inter rater agreement (IRA) for all dimensions of the survey. We followed the suggestions of (Gwet, 2012) for the selection of agreement coefficients and weights. The raters show substantial agreement regarding the query complexity, almost perfect agreement regarding plausibility and typicality, and moderate agreement regarding their preference of the NLPS, as shown in table 3.

Table 4: Performance scores for the NLPS variants - numbers in bold indicate the preferable NLPS variant according to the score of the corresponding metric.

NLPS	Rater Preference	∅ Characters	∅ Term Frequency	∅ Flesch Reading Ease	∅ Cosine Similarity to Query	Passed Quality Check	∅ Dependency Distance	∅ Coherence
Baseline	289	459	31	45	**63%**	138	**2.4**	80%
Hybrid	**378**	**225**	**26**	**65**	56%	**478**	3.5	**87%**

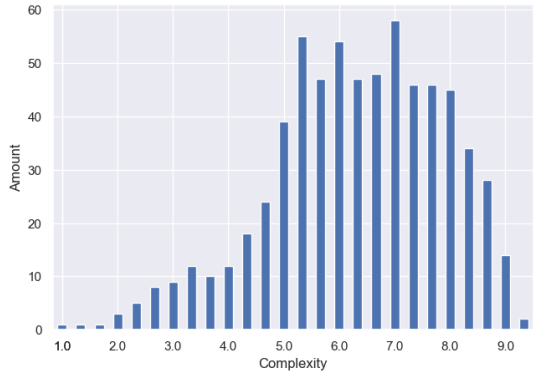


Figure 7: Distribution of the average complexity rating.

5.2.2 Perceived Complexity

Figure 7 shows the distribution of the average complexity rating per SQL-query. Most of the queries are rated to be of moderate to high complexity on average. In 20 cases, all raters agree on the same complexity value. In 214 cases, two raters agree on the same complexity value. In the remaining 433 cases, there is no agreement between raters. The overall observed agreement is 13.7%. In the case of no agreement between raters, the average complexity rating is 6.25 with a standard deviation of 2.80. In the case of some agreement between raters, the average complexity rating is 6.38 with a standard deviation of 1.97. When each complexity rating is sorted into equal-width bins (easy: 1-3, medium: 4-6, hard: 7-10), the count of agreement between two raters increases to 433. The count of agreement between three raters increases to 144 and the count of absolute disagreement between raters decreases to 90. In those 90 cases, the average complexity rating is 5.51 with a standard deviation of 3.7.

5.2.3 Plausibility and Typicality of the Generated SQL-Queries

Figure 8 shows the distribution of the majority rating, as described in section 5.1.1, for plausibility and typicality per SQL-query. The solid bars show the distribution of majority ratings according to strategy two for dissolving inconclusive ratings, and the dashed bars show the distribution according to majority ratings of strategy 3 for dissolving inconclusive ratings. Most of the queries are rated somewhat or very plausible and typical, a moderate amount is rated neutral and very little queries are rated to be somewhat implausible or atypical. The overall observed agreement is 33.2% regarding plausibility and 30.1% regarding typicality. The average rating regarding plausibility is 0.98 and the standard deviation is 0.77. The average rating regarding typicality is 0.86, the standard deviation is 0.81.

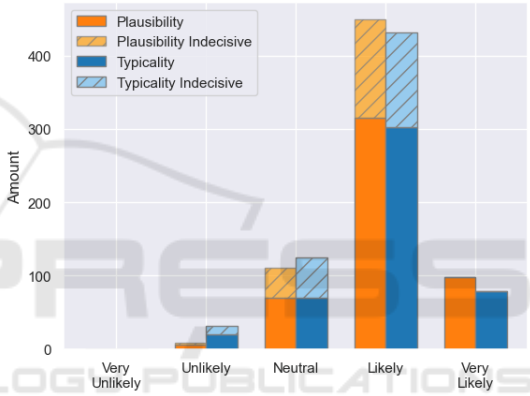


Figure 8: Distribution of plausibility and typicality ratings.

5.2.4 Human-Likeness of the Generated NLPS

Table 4 shows the rating result and the performance of the two variants of the NLPS regarding the metrics specified in section 5.1.3. Of the two generated NLPS, the hybrid was preferred by a majority vote, according to strategy two described in section 5.1.1, in 378 cases, the baseline was preferred in the remaining 289 cases. The rounded average character count is 225 for the hybrid approach and 459 for the baseline. The rounded average term frequency is 26 for and 31 respectively. The rounded average Flesch Reading Ease score is 65 and 45. The rounded average semantic similarity to the SQL-query is 56% and 63%. 478 out of 667 statements, and 138 out of 667 statements pass the quality test for the respective approach. The rounded average dependency distance is 3.5 and 2.4. The average coherence is 87% and 80%.

[4] github.com/HTW-ALADIN/SQL-Query-Generation
[5] Available at https://datasets.imdbws.com/

5.3 Discussion of the Results

The results show that the presented approach for the generation of SQL-query items produces SQL-queries with adequate diversity regarding their content and complexity, but raters often disagree regarding the concrete complexity, even when binned into more coarse categories. Interestingly, no major differences between the composition of the SQL-queries could be found between the 90 cases of strong disagreement and the remaining cases. Thus, the subjective perception of SQL-query item complexity should be researched further by taking learners and verified experts into account. The raters consider most of the queries semantically plausible and typical. Thus, requirement *R4.1* and *R4.2* is considered to be fulfilled. The rater preference for the NLPS generated by the hybrid approach was only $\approx 57\%$, even though most other applied metrics favor the hybrid approach significantly. The preferences and effects of NLPS alternatives should be further studied, especially regarding their influence on the complexity of SQL-query items.

Possible threats to the validity of the study may be rater subjectivity and lack of expertise. The unclear NLPS preference of the raters may be due to personal stylistic preferences and might require a larger pool of raters for increasing validity. While the raters were selected through a pretest screening, they may not match the knowledge of SQL experts. The study has limited generalizability as the evaluation data set was generated in the context of one database schema and thus may not represent the diversity encountered in real-world scenarios.

6 CONCLUSION

SQL is an important skill to master for computer science students and database professionals. Attaining a high level of mastery for SQL requires practicing with a large amount of SQL-query items. Since the manual production of such items is expensive and labor-intensive, automatic generation is preferred. The automatic generation of SQL-query items should allow for its parameterization to generate SQL-queries with varying degrees of complexity and diverse content. The generated SQL-queries should be semantically plausible. The accompanying NLPS should be human-like and unambigous. Current approaches for the generation of SQL-query items do not fulfill the aforementioned criteria.

This paper proposed an approach and evaluated it against multiple requirements. The approach constructs a knowledge graph that reflects the properties of the database schema and infers constraints for the generation of query constituents. These contraints enable the creation of meaningful queries, and thus fulfills *R4.1* and *R4.2*. Based on the generated queries an NLPS is generated. By utilizing MASK-filling, the human effort for the surface realization is reduced compared to prior approaches. The generation process supports a wider variety of SQL keywords compared to prior apporaches and is easily extendable.

The evaluation used a set of generated SQL-query items each rated by three raters and additional metrics. Survey results show that the majority of items were considered plausible and typical. While several metrics were infavour of the hybrid approach of NLPS generation raters did not show a clear preference. Using the input parameters, SQL queries with a high diversity of SQL consitutents could be generated (*R2*) that were deemed to adress the full spectrum of complexity (*R1*).

While our approach is capable of generating items fullfilling the requirements the quality of the SQL-queries, the requirements regarding the formulation of the NLPS shows room for improvement. Therefore, approaches for the generation of NLPS could be compared to NLPS purely generated by LLMs and evaluated regarding their unambiguity. Further research is required to develop more precise metrics for measuring the complexity of SQL-query items, which is essential for developing adaptive learning systems. Future evaluation should include human expert raters, to judge SQL-specific as well as conventional item quality metrics for the generated items.

Research on reducing the human effort required for the schema annotation and allowing for the personalization of SQL-query items to learner preferences can build on our results, e.g., by including schema generation in SQL-item generation.

To further reduce human effort in the assessment process for SQL-query items, research on the automatic grading and generation of feedback for SQL-query items may be conducted. Research on such approaches may benefit from building on top of the proposed approach, as the generated items contain more information, as could be infered from manually generated items.

REFERENCES

Aerts, W., Fletcher, G., and Miedema, D. (2024). A feasibility study on automated sql exercise generation with chatgpt-3.5. In *Proceedings of the 3rd International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, DataEd '24,

page 13–19, New York, NY, USA. Association for Computing Machinery.

Atchariyachanvanich, K., Nalintippayawong, S., and Julavanich, T. (2019). Reverse SQL Question Generation Algorithm in the DBLearn Adaptive E-Learning System. *IEEE Access*, 7:54993–55004.

Becker, M., Liang, S., and Frank, A. (2021). Reconstructing Implicit Knowledge with Language Models. In *Workshop on Knowledge Extraction and Integration for Deep Learning Architectures; Deep Learning Inside Out*.

Cc2020 Task Force (2020). *Computing Curricula 2020: Paradigms for Global Computing Education*. ACM, New York, NY, USA.

Chaudhari, M. S., Hire, M. A., Mandale, M. B., and Vanjari, M. S. (2021). Structural Query Language Question Creation by using Inverse Way.

Do, Q. C. D., Agrawal, R., Rao, D., and Gudivada, V. N. (2014). Automatic Generation of SQL Queries.

Eleftherakis, S., Gkini, O., and Koutrika, G. (2021). Let the Database Talk Back: Natural Language Explanations for SQL. In *SEA-DataVLDB*.

Elmasri, R. and Navathe, S. (2016). *Fundamentals of database systems*. Pearson, Hoboken, NJ, seventh edition edition.

Ericsson, K. A. (2006). The Influence of Experience and Deliberate Practice on the Development of Superior Expert Performance. In *The Cambridge Handbook of Expertise and Expert Performance*, Cambridge Handbooks in Psychology, pages 683–704. Cambridge University Press, Cambridge.

Farahnak, F., Rafiee, L., Kosseim, L., and Fevens, T. (2020). Surface realization using pretrained language models. In *IEEE Working Conference on Mining Software Repositories*.

Flesch, R. F. (1948). A new readability yardstick. *Journal of Applied Psychology*, 32(3):221–233.

Gudivada, V. N., Arbabifard, K., and Rao, D. (2017). Automated Generation of SQL Queries that Feature Specified SQL Constructs.

Gwet, K. (2012). *Handbook of inter-rater reliability: The definitive guide to measuring the extent of agreement among raters*.

Hauser, F., Stark, T., Mottok, J., Gruber, H., and Reuter, R. (2020). Deliberate Practice in Programming: How is it carried out by programmers? In *Proceedings of the 4th European Conference on Software Engineering Education*, ECSEE '20, pages 42–46, New York, NY, USA. Association for Computing Machinery.

ISO Central Secretary (2016). Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation). Standard ISO/IEC 9075-2:2016/Cor 2:2022, International Organization for Standardization.

Iyer, S., Konstas, I., Cheung, A., and Zettlemoyer, L. (2016). Summarizing Source Code using a Neural Attention Model. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Kokkalis, A., Vagenas, P., Zervakis, A., Simitsis, A., Koutrika, G., and Ioannidis, Y. E. (2012). Logos: a system for translating queries into narratives. *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*.

Kosh, A., Simpson, M. A., Bickel, L., Kellogg, M., and Sanford-Moore, E. (2018). A Cost-Benefit Analysis of Automatic Item Generation. *Educational Measurement: Issues and Practice*, 38.

Koutrika, G., Simitsis, A., and Ioannidis, Y. E. (2010). Explaining structured queries in natural language. *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 333–344.

Krippendorff, K. (2004). Reliability in content analysis: Some common misconceptions and recommendations. *Human Communication Research*, 30:411–433.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

Li, Y., Radev, D., and Rafiei, D. (2024). *Natural Language Interfaces to Databases*. Synthesis Lectures on Data Management. Springer International Publishing, Cham.

Ma, D., Chen, X., Cao, R., Chen, Z., Chen, L., and Yu, K. (2021). Relation-Aware Graph Transformer for SQL-to-Text Generation. *Applied Sciences*.

Mandamadiotis, A., Koutrika, G., and Amer-Yahia, S. (2024). Guided sql-based data exploration with user feedback. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 4884–4896.

Oya, M. (2011). Syntactic dependency distance as sentence complexity measure. *Proceedings of the 16th International Conference of Pan-Pacific Association of Applied Linguistics*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2023). Exploring the limits of transfer learning with a unified text-to-text transformer.

Raslan, G. The Impact of the Zone of Proximal Development Concept (Scaffolding) on the Students Problem Solving Skills and Learning Outcomes. In *BUiD Doctoral Research Conference 2023*, pages 59–66, Cham. Springer Nature Switzerland.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks.

Stephen Cass (2024). Top Programming Languages 2024.

van der Lee, C., Gatt, A., van Miltenburg, E., and Krahmer, E. J. (2021). Human evaluation of automatically generated text: Current trends and best practice guidelines. *Comput. Speech Lang.*, 67:101151.

Westacott, R., Badger, K., Kluth, D., Gurnell, M., Reed, M. W. R., and Sam, A. H. (2023). Automated Item Generation: impact of item variants on performance and standard setting. *BMC Medical Education*, 23(1):659.

Xu, K., Wu, L., Wang, Z., Yu, M., Chen, L., and Sheinin, V. (2018). SQL-to-Text Generation with Graph-to-Sequence Model. In *Conference on Empirical Methods in Natural Language Processing*.