Automated Generation of SQL Queries that Feature Specified SQL Constructs

Venkat N Gudivada*, Kamyar Arbabifard*, and Dhana Rao†
*Department of Computer Science, East Carolina University, USA
†Department of Biology, East Carolina University, USA
email: gudivadav15@ecu.edu, arbabifardk15@students.ecu.edu, and raodh16@ecu.edu

Abstract – SQL is an ISO standard language for querying relational databases. SQL queries are deceptively simple to write, but writing semantically correct queries requires a good understanding of the data model and SQL constructs. Often, this is a challenging task for beginners. Automatic generation of SQL queries that feature specified SQL constructs is useful for both informal self-testing and formal assessment. In this work-in-progress paper, we describe the automated question generation problem in a broader context, provide an overview of the current approaches, and discuss our approach to automatic generation of SQL queries. Our approach is based on the notion of *grammar graph*. We illustrate the approach using an arithmetic expression grammar and generalize this approach to SQL query generation.

Keywords—Context-Free Grammar*; *SQL Query Generation*; *Grammar Graph*; *Question Generation.

I. Context and Introduction

Recently, there has been tremendous interest in improving student learning in classrooms through innovative and inclusive pedagogy. Research in cognitive psychology, neuroscience, and biology provides insight into how humans learn [1]. Contrary to the conventional study habits such as cramming, re-reading, and single-minded repetition, techniques such as interleaving the practice of one skill or topic with another, and self-testing enable more complex and durable learning outcomes [2]. There is research-based evidence for seven key aspects of learning including effect of prior knowledge, organizing knowledge to effect learning, factors that motivate students, process by which students develop mastery, self-testing, kinds of practices and feedback that enhance learning, and the processes by which students become self-directed learners [3]. Several strategies exist to bring learning research into classroom environments across diverse disciplines [4] [5] [6]. In this paper, our focus is on enhancing learning through self-testing.

The National Academy of Engineering of The National Academies has identified advancing personalized learning as one of the fourteen Grand Challenges for Engineering in the 21st century [7]. Personalized learning has multiple dimensions. Providing a wide assortment of teaching and learning materials to suit the different learning styles of students is one aspect. Allowing students to progress through a course to meet their just-in-time learning goals is another aspect. More specifically, each student may potentially choose a different order for learning the course topics. The only constraints that limit the topic order are the prerequisite dependencies. A third aspect of personalization involves providing contextualized scaffolding and immediate feedback to students on assessment activities. The last aspect involves providing students authentic questions for self-assessment and preparation for exams.

Structured Query Language (SQL) is an ANSI and ISO standard declarative query language for querying and manipulating relational databases. Though writing SQL queries appears to be easy at a superficial level, students tend to make several types of errors [8]. The goal of this paper is to provide a question generation tool that automatically generates virtually unlimited SQL queries to support personalized learning in a database systems course. The tool will generate SQL queries that will contain the SQL constructs specified by the user. Given the complexity of the SQL language, we begin our investigation of automated question generation on a simpler problem: generation of arithmetic expressions. Once we understand the generation process and formalize an algorithm, we apply the algorithm to the generation of SQL queries.

In Section II, we provide motivation for automated question generation problem in a broader context and discuss related work. An automated approach to arithmetic expression generation is described in Section III. Extending this work to SQL query generation is outlined in Section IV. Section V provides conclusions.

II. Motivation and Related Work

The advent of Massive Open Online Courses (MOOCs), renewed interest in anytime and anywhere learning, the potential of personalization in revolutionizing learning, benefits of contextualized scaffolding, and automated and immediate feedback on learning assessments are the primary drivers for automated question generation. This is an area of interest to researchers across multiple disciplines. Approaches to automated question generation are as diverse as the disciplines themselves. Furthermore, the goal of question generation is not necessarily for learning assessment.

We categorize current approaches to question generation into three broad categories: template-based, Natural Language Processing (NLP) based, and hybrid. Template-based approaches use knowledge structures such as ontologies and manually crafted templates. NLP-based approaches analyze natural language text for extracting semantic information and use that information for question generation. As the name implies, hybrid approaches draw upon templates and NLP techniques.

A. Template-based Approaches

Khalek and Khurshid present an approach to generating syntactically and semantically correct SQL queries [9]. The context for their investigation is testing of relational database engines. They translate the problem of generating SQL queries into a Satisfiability (SAT) problem. More specifically, they translate SQL query constraints into Alloy models, which generate SQL queries. They also generate data to populate databases and test database engines using the generated SQL queries.

Binnig et al. propose an approach to *query-aware* database for testing a Database Management System (DBMS) [10]. The purpose of this research is to ensure the availability of appropriate data in the database to enable matching the data returned by a query to the expected result. If the database does not contain the appropriate data, the query will execute but does not return any results.

An integrated Exploratorium for database courses is developed as a platform to investigate the technical problems and the pedagogical benefits of using diverse interactive learning tools in [11]. It provides personalized access to three types of interactive learning tools: annotated examples, self-assessment questions, and SQL lab. Over 400 self-assessment SQL questions are generated from 50 templates.

Do et al. propose an approach to SQL query generation using manually crafted templates and SQL ontology [12]. A major limitation of this approach is that the generated queries are limited to the pre-defined templates. Another approach to testing database applications using automatically generated test cases is discussed in [13].

Siddiqi et al. describe the development and evaluation of IndusMarker, a short-answer grading system for an object-oriented programming course [14]. IndusMarker targets factual answers and uses *structure matching* for determining correctness of students' responses. Lastly, Li and Sambasivam developed a *template-based* approach to automated question generation for intelligent tutoring applications [15, 16]. Template-based approaches are also used to generate both questions and expected answers to evaluate retrieval algorithms for unstructured data [17].

B. NLP-based Approaches

Automatic generation of factual *WH*-questions from texts with potential educational value is investigated in [18]. WH-questions are those that contain an interrogative pro-form. For example, words that begin wh-questions are who, what, when, where, why, and how. An automated system is developed for automated question generation, which uses natural language processing techniques, manually encoded transformation rules, and a trained statistical question ranker.

To assist the task of automatically assigning texts for students to read, vocabulary assessment must be performed first. A system for vocabulary assessment is discussed in [19]. It generates six types of vocabulary questions using WordNet data. Evaluation of the system performance indicates that the vocabulary skill measured using the generated questions correlates well with the same measured on independently developed human written questions. An extension of this system for the Portuguese context is discussed in [20]. Another approach to language learning and assessment is discussed in [21]. This system semi-automatically generates questions for testing grammar knowledge using manually-designed patterns and natural language processing techniques.

Three workshops were held on question generation [22]. The third workshop on Question Generation included a question generation shared task and evaluation challenge, which featured question generation from sentences and question generation from paragraphs [23]. A special issue of Dialog & Discourse journal featured NLP-based question generation topics [24].

C. Hybrid Approaches

Ontologies are knowledge structures which depict entities in a domain and relationships among the entities. Automated inference and reasoning were the two primary and original drivers for ontologies. Al-Yahya developed a system for multiple choice question (MCQ) generation using ontologies [25]. The system is called *OntoQue* and has been evaluated on two domain ontologies. The evaluation findings indicated a limited success of the approach and revealed a number of shortcomings from the perspective of educational significance of MCQs. This study also suggests a holistic approach, which incorporates learning objectives and content, lexical knowledge, and scenarios into a single cohesive framework.

III. Generation of Arithmetic Expressions

Our approach to question generation is based on Context-Free Grammars (CFG). Conceptually, a CFG is specified by a set of rules or productions. The use of CFG to describe grammars of natural languages traces back to Panini (6th - 4th century BCE), a Sanskrit grammarian. The mathematical formalism of CFGs was developed by Noam Chomsky in mid 1950s. CFGs became a standard formalism for describing the grammars of computer programming languages in late 1950s. A *parser* is a computer program which determines, given a string and a CGF, whether the string can be generated from the CFG. In other words, the parser determines whether or not the string is valid element in the language defined by the CFG.

TABLE I. A CONTEXT-FREE GRAMMAR (CFG) FOR ARITHMETIC EXPRESSIONS

<expr>	::= <term> [<expr1>]*
<expr1>	::= (+ | -) <term>
<term>	::= <factor> [<expr2>]*
<factor>	::= <base> [<expr3>]*
<base>	::= (<expr>) | <number>
<expr2>	::= (* | /) <factor>
<expr3>	::= ∧ <exponent>
<exponent>	::= (<expr>) | <number>
<number>	::= <digit> [<digit>]*
<digit>	::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Efficient parsers exist to determine whether a program written a programming language such as Java conforms to Java CFG.

Though the CFG for a programming language is finite, one can generate an infinite number of programs in the language. However, these programs are manually constructed by programmers. Writing useful programs is an intellectual task and requires knowledge and skill. The problem we address in this paper is the automatic generation of strings from a given CFG, which contain user specified *keywords* or *constructs*. The latter are *literals* in the CFG. This problem is challenging because strings with certain combinations of keywords may not exist in the language defined by the CFG. The first step is to determine whether a string that contains the user specified keywords exists. If such a string exists, we then generate it. We demonstrate our approach on a simple grammar first and then generalize the approach to SQL query generation.

The CFG we use for generating arithmetic expressions is shown in Table I. Using CFG, we generate arithmetic expressions of arbitrary complexity. Operands in the expressions are integers and operators include addition (+), subtraction (-), multiplication (*), division (/), and exponentiation (∧).

We propose a graph-based representation for efficiently generating arithmetic expressions from CFG grammars. We refer to this as the *grammar graph* and is shown in Figure 1. This is similar to Deterministic Finite State Automata (DFSA) but the semantics are different. The grammar graph consists of a set of vertices and edges. The color coding, line types, and other markers capture critical information to aid the generation of arithmetic expressions. Each vertex in the graph corresponds to a *terminal* or *non-terminal* in the grammar. In Figure 1, there is only one terminal designated by the vertex labeled <digit>. Note the color of the vertex.

The graph node labeled <expr> is the start vertex for expression generation. The directed edge from <expr> to <term> indicates a *substitution* — the non-terminal <expr> is replaced by another nonterminal <term>. Next, consider the *red-dashed* directed edge from <term> to <expr1>. This denotes an *optional edge* and does not involve replacing <term> with <expr1>.

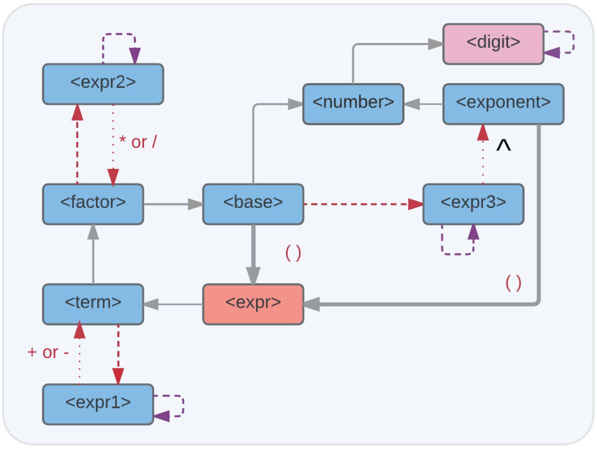

Figure 1. Graph representation of a Context-Free Grammar

The optional edge semantic is that whatever is generated through the optional edge gets appended to the <term>. In other words, instead of replacement something gets appended to the <term>.

The loop on the vertex labeled <expr1> denotes that zero or more copies of <expr1> are appended to <expr1>. Next, consider the *red-dotted* directional edge from <expr1> to <term>. Notice the edge label: plus (+) or minus (-). The semantic is that each copy of <expr1> is replaced by prefixing <term> with plus (+) or minus (-). For example, <expr1> can be replaced by either + <term> or - <term>. Lastly, consider the *thick-lined* directed edge from <base> to <expr> and notice the edge label: ( ). The semantic of this notation is that <base> is replaced by <expr> enclosed in parenthesis. That is, <base> is replaced by (<expr>).

Consider generating an arithmetic expression of the form: 32 + 65 − 173. As noted earlier, the generation process always starts at the vertex named <expr>. Next, since there is an edge from <expr> to <term>, we replace <expr> by <term>. Traversal of the edge from <term> to <expr1> is optional. If this path is chosen, we append to <term> rather than replacing it. We choose this optional edge and visit <expr1>. The loop indicates zero or more repetitions and each repetition generates one <expr1>. Let us generate two copies – <expr1> <expr1>. Next, consider the edge from <expr1> to <term>. Each copy of <expr1> will be replaced by a plus (+) or a minus (-) followed by the <term>. Assume that we chose plus in the first case and minus in the second case. Now we have the string <term> + <term> - <term>. Next, using the edge from <term> to <factor>, each copy of <term> is replaced by <factor> yielding <factor> + <factor> - <factor>. Similarly, we traverse from <factor> to <base> yielding <base> + <base> - <base>. Repeating this one more time using the edge from <base> to <number>, we get <number>

+ <number> - <number>. Using the <number> – <digit> directed edge and looping on the <digit>, each <number> in <number> + <number> - <number> can be replaced by a desired integer number, which yields 32 + 65 - 173. Using a similar procedure, we can generate any number of arbitrarily complex arithmetic expressions such as 9∧((8*9)∧5*(8*((5*(7∧(09/95)/9))+(9-(4∧(((6∧9)+2)+((81/877)∧5)∧9)))+8)∧3+8))/8.

In the grammar graph, we distinguish between two types of paths: simple and complex. All paths start at the special vertex <expr> and end at a terminal vertex (e.g., <digit>). A *simple path* is one that does not involve any loops or optional edge traversals. For example, <expr> → <term> → <factor> → <base> → <number> → <digit> is a simple path. Simple path traversals yield simplest arithmetic expressions such as 4 and 6. *Complex paths* are generated from simple paths by adding optional traversals, single- and multi-vertex loops. For instance, adding a single vertex loop, we can generate expressions such as 15 and 5674. An example of a multi-vertex loop is <expr> → <term> → <factor> → <base> → <expr>.

In our expression generation algorithm, a user can specify the complexity of the generated arithmetic expression. An user may use the terms *simple*, *moderate*, and *complex* to denote expression complexity. The algorithm quantifies the level of complexity using the length and the number of operators in the expression generated.

Our goal is to generate expressions of arbitrary complexity, which feature specified arithmetic operators from the set {add, subtract, multiply, divide, exp}. Given the size of the grammar, this task is not complex. As the size and complexity of the grammar increases, generating a query that features given operators is non-trivial. We address these issues in the context of generating SQL queries that contain specified SQL constructs.

IV. Generating SQL Queries

ISO/IEC 9075:2011 is the standard for the SQL database query language. The SQL language elements include operators, clauses, predicates, expressions, statements, and queries. Operators include the traditional mathematical ones such as ≤ and >, as well as database-specific ones such as Between, In, Exists, Is Not Distinct From, and Avg. Clauses are components of statements and queries. Predicates are conditions that evaluate to three-valued logic (true, false, unknown). Expressions, when evaluated, produce either scalar values or tables. Statements enable specifying a wide range of actions on the database. Lastly, queries enable retrieving data from the database. Queries do not change database contents and operate in read-only mode. SQL queries comprise a principal component of the ISO/IEC standard. Table II shows the generic structure of SQL queries. Only the first two components are mandatory. However, the components must occur in the order indicated. For example, a SQL query may have Select, From, and Group By without the Where clause. Likewise, we can have Select, From, and Order By without the Group By and Having clauses.

Table II. General structure of SQL queries

Select	<column names and transformations on column values>
From	<table names> and <join conditions>
Where	<row restrictions>
Group By	<column names for grouping rows in the result set>
Having	<condition specifying which groups to keep>
Order By	<sorting specification for displaying results>

Drawing upon our experience in generating arithmetic expressions, we will first create a grammar graph using the SQL grammar. We subset the SQL grammar to include only rules that are associated with the Select statement. Next, using the grammar graph we will determine if it is feasible to generate a query which contains the user-specified SQL constructs. This requires determining a path in the graph which encompasses, starting at the vertex which corresponds to the start symbol of the grammar (e.g., <expr)> in Figure 1), vertices and edges corresponding to all the SQL constructs specified by the user. Furthermore, the string traced by this path either contains only terminals or non-terminals that can be replaced with terminals.

V. Conclusions and Future Work

In this work-in-progress paper, we have presented a novel approach to automatic generation of SQL queries that feature user-specified SQL constructs. Our approach uses the notion of *grammar graph* to determine whether or not such a query exists, and to generate the query. We have demonstrated the validity of the approach on arithmetic expression generation. As a logical next step, we will apply the approach to the actual generation of SQL queries.

References

[1] L. D. Fink, *Creating significant learning experiences: An integrated approach to designing college courses*, Second. San Francisco, CA: Jossey-Bass, 2013.

[2] P. C. Brown, H. L. Roediger, and M. A. McDaniel, *Make it stick: The science of successful learning.* Cambridge, MA: Belknap Press, 2014.

[3] S. A. Ambrose, M. W. Bridges, M. DiPietro, M. C. Lovett, and M. K. Norman, *How learning works: Seven research-based principles for smart teaching.* San Francisco, CA: Jossey-Bass, 2010.

[4] V. Gudivada, J. Nandigam, and D. Guru, “A learning-centered approach to designing computer science courses,” *J. Comput. Small Coll.*, vol. 21, no. 4, pp. 96–103, Apr. 2006.

[5] J. M. Lang, *Small teaching: Everyday lessons from the science of learning*, First. San Francisco, CA: Jossey-Bass, 2016.

[6] M. Weimer, *Learner-centered teaching: Five key changes to practice*, Second. San Francisco, CA: Jossey-Bass, 2013.

[7] National Academy of Engineering. (2017). Grand challenges, [Online]. Available: http : / / www . engineeringchallenges . org / cms / challenges . aspx (visited on 02/08/2017).

[8] A. Ahadi, V. Behbood, A. Vihavainen, J. Prior, and R. Lister, "Students' syntactic mistakes in writing seven different types of sql queries and its application to predicting students' success," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, ser. SIGCSE '16, Memphis, Tennessee, USA: ACM, 2016, pp. 401–406.

[9] A. S. Khalek and S. Khurshid, "Automated sql query generation for systematic testing of database engines," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ser. ASE '10, New York, NY: ACM, 2010, pp. 329–332.

[10] C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu, "Qagen: Generating query-aware test databases," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '07, New York, NY: ACM, 2007, pp. 341–352.

[11] P. Brusilovsky, S. Sosnovsky, M. V. Yudelson, D. H. Lee, V. Zadorozhny, and X. Zhou, "Learning sql programming with interactive tools: From integration to personalization," *Trans. Comput. Educ.*, vol. 9, no. 4, 19:1–19:15, Jan. 2010.

[12] Q. Do, R. Agrawal, D. Rao, and V. Gudivada, "Automatic generation of sql queries," in *Proceedings of the* 121^{st} *ASEE Annual Conference & Exposition*, ASEE, Jun. 2014, pp. 1–11.

[13] D. Chays, "Test data generation for relational database applications," AAI3115007, PhD thesis, Brooklyn, NY, 2004.

[14] R. Siddiqi, C. J. Harrison, and R. Siddiqi, "Improving teaching and learning through automated short-answer marking," *IEEE Transactions on Learning Technologies*, vol. 3, pp. 237–249, 2010.

[15] T. Li and S. Sambasivam, "Question difficulty assessment in intelligent tutor systems for computer architecture," in *Proc ISECON*, EDSIG, 2003, pp. 1–8.

[16] T. Li and S. Sambasivam, "Automatically generating questions in multiple variables for intelligent tutoring," in *Society for Information Technology & Teacher Education International Conference (SITE)*, 2005, pp. 471–2005.

[17] V. Gudivada, "$TESSA$ - an image testbed for evaluating 2-D spatial similarity algorithms," *ACM SIGIR Forum*, vol. 28, no. 2, pp. 17–36, Fall 1994.

[18] M. Heilman, "Automatic factual question generation from text," PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2011.

[19] J. C. Brown, G. A. Frishkoff, and M. Eskenazi, "Automatic question generation for vocabulary assessment," in *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, Morristown, NJ: Association for Computational Linguistics, 2005, pp. 819–826.

[20] R. Correia, "Automatic question generation for REAP.PT tutoring system," Master's thesis, 2010. [Online]. Available: http : / / www . inesc - id . pt / pt / indicadores / Ficheiros / 5599 . pdf (visited on 01/15/2017).

[21] C.-Y. Chen, H.-C. Liou, J. S. Chang, and J. S. Chang, "FAST – an automatic generation system for grammar tests," in *Proceedings of the COLING/ACL on Interactive presentation sessions*, 2006, pp. 1–4.

[22] P. Piwek and K. Boyer. (2010). The third workshop on question generation, [Online]. Available: http://oro.open.ac.uk/22343/1/QG2010-Proceedings.pdf (visited on 01/15/2017).

[23] V. Rus, P. Piwek, S. Stoyanchev, B. Wyse, M. Lintean, and C. Moldovan, "Question generation shared task and evaluation challenge: Status report," in *Proceedings of the* 13^{th} *European Workshop on Natural Language Generation*, ser. ENLG '11, Stroudsburg, PA: Association for Computational Linguistics, 2011, pp. 318–320.

[24] P. Piwek and K. Boyer, "Special issue on question generation," *Dialog & Discourse*, vol. 3, pp. 1–322, 2012, http://elanguage.net/journals/dad/issue/view/347.

[25] M. Al-Yahya, "Ontology-based multiple choice question generation," *The Scientific World Journal*, no. 10.1155/2014/274949, pp. 1–9, 2014.