Using Context Free Grammars to Generate SQL Questions

Jared Conway* and Nada Basit

ABSTRACT

This paper showcases the limitations of current SQL practice approaches, both in server architecture and implementation. The technical project works towards resolving these limitations by providing a fully client-sided website to help students practice SQL. By providing a client only architecture rather than the traditional server-client approach, queries are guaranteed to be sandboxed, and the necessity for reliable WiFi becomes nearly nonexistent. This is in part an extension of the SQL Injection platform first presented in 2019 [1], with more focus on practicing SQL queries than SQL injection attacks. Such practice involves basic keywords, including WHERE, HAVING, and GROUP BY as well as aggregates. This approach implements modified context free grammars for constructing questions and example answers, including variables for aggregates and GROUP BY.

CCS CONCEPTS

• **Theory of computation → Formal languages and automata theory; • Applied computing → Education;**

KEYWORDS

SQL; education; context free grammar; question generation

1 INTRODUCTION

Experiential learning, otherwise known as “learn-by-doing,” is defined as the “sense-making process of active engagement between the inner world of the person and the outer world of the environment” [2]. Some schools, such as UVA, even have sections that talk more about how their curriculums offer experiential learning due to how effective of a learning strategy it is [3].

One such concept that is particularly useful to learn using experiential learning is SQL, or Structured Query Language. SQL is used in software development whenever large amounts of data needs to be stored, updated, read, or deleted. Many companies, such as Oracle, rely on SQL during their day-to-day operations, so knowledge of SQL can be vital when joining the workforce.

Due to SQL being a programming language, practice can be done with a computer instead of on paper. Approaches which use computers have the benefit of being more extendable and reusable than paper methods. Instead of instructors using paper and ink to print physical assignments for each student, students can instead use their electronic devices to access and complete online assignments. Additionally, instead of printing a page for every new assignment, the instructor only needs to add a new section to the already existing website. Such assignments may even have an autograde feature, allowing students to check the correctness of their SQL queries in real time.

2 RELATED WORK

Many websites have been able to leverage these benefits in order to help people learn SQL. One such website is the SQL Injection platform this technical project extends. That platform implements a server-client architecture which allows users to submit SQL queries which exploit the website’s vulnerabilities and inject malicious SQL. The intention of the platform is to show students how bad actors can bypass their security features, as well as underscoring the importance of always checking user queries [1]. One downside, however, is that some assignments require an administrator to reset the database after usage, meaning it is unable to be fully reusable.

An alternative approach which resolves this issue is to use a regex, or regular expression, instead of allowing the user to run a real SQL query. Typically, these approaches have a set of pre-made questions and ask the user to fill in a blank to complete the query, w3schools being one such website [4]. After the user fills in the blank, the system checks to see if their answer matches the correct answer; if

so, the user is right, and if not, the user is wrong. Although sites like this do offer an environment that allows users to run SQL queries, the exercises do not take advantage of these environments. These sites are tailored more towards people new to SQL, but not those who already understand keywords and wish to consolidate what they already know. Additionally, as SQL is a programming language, there is not necessarily always one correct answer for any query; therefore, a simple regex approach would only be able to check one correct answer, but not alternatives which return the same query.

3 OVERVIEW

In order to have an approach which is both extendible and reusable, we present our own website which is capable of generating SQL practice questions, adapting context free grammars with constructing questions and answers. Similar to the Injection platform, our website allows users to input any SQL query into a textbox to be run and checked. Unlike the platform, this query is run on the client rather than on the server, meaning an administrator does not need to reset a database every time someone drops a table. Besides sending the client HTML and Javascript, there is no backend required for this approach. A pleasant side effect of using a frontend only architecture is that users are capable of running queries on unreliable or nonexistent WiFi, assuming their internet browser is capable of caching the webpage, as there is nothing to send to the server.

Figure 1: Example Generated SQL Question and Answer. The user is able to type SQL in the bottom box, and click "Run" once they are finished.

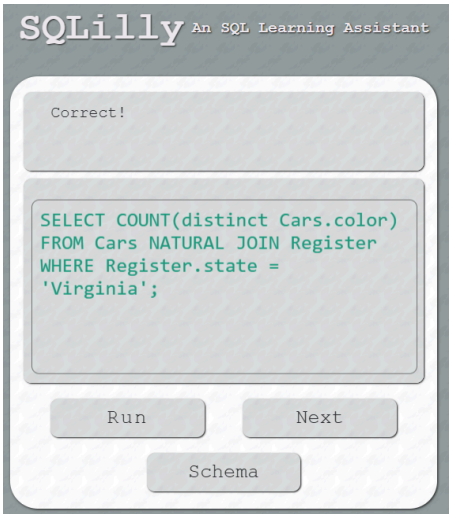

Figure 2: Correct SQL Query. The question is replaced with the word "Correct!" and the user's query flashes green.

This approach has additional benefits over the regex approach as its implementation is capable of accounting for all valid queries instead of just one. Instead of comparing the input values, our implementation compares the outputs the generated answer produces with the submitted answer. By comparing outputs instead of inputs, our system is able to treat any query submitted by the user as a black box - as long as the query has the correct behavior, it does not matter what goes on "under the hood." This allows users to use more advanced methods, such as subquerying, to see how different approaches to the same problem can be performed.

4 IMPLEMENTATION

The approach is based on context free grammars, with the addition of a variable $\mathbf{x_1}$. Both questions and answers have similar, but distinct grammars. Let ε denote an empty string.

4.1 QUESTION GRAMMAR

Each question generated using this approach is of the form:

```
S → "Find" [A | B [C | ε][D | ε]] "."
A → [["all distinct" | "the amount of distinct"]
    W [C | ε] | "the amount of each distinct" W
    [C | ε][D | ε]]]
B → "the" X Z "of each" Z
C → "whose" Z ["is not" | "is" | "is less than"
    | "is greater than"] [value] ["and" C | ε]
D → ", having its" X Y be at least 0
W → [[table name] | Z]
X → ["average" | "minimum" | "maximum"]
Y → [table name] [column name (integer only)]
Z → [table name] [column name]
```

4.2 ANSWER GRAMMAR

Each example answer generated using this approach is of the form:

```
S → “SELECT” [T | U] “;”
T → [“DISTINCT” Z A | [“COUNT(DISTINCT” Z “)”
    A | “DISTINCT” x1 “, COUNT(*)” B]]
U → [x1 “,” V B | V A]
V → [“AVG(” | “MIN(” | “MAX(”]  Z “)”
A → “FROM Cars NATURAL JOIN Register”
    (“WHERE” C | ε)
B → A “GROUP BY” x1 (“HAVING” D | ε)
C → Z (“!=” | ”==” | ”<” | ”>” ) [value]
    (“ and” C | ε)
D → X “> 0”
X → [“AVG(” Y “)” | “MIN(” Y “)” | “MAX(” Y “)”]
Y → [table name].[column name (integer only)]
Z → [table name].[column name]
```

4.3 QUERY VALIDATION

After generating the question and example answer, the user is given the question, and the answer query is run in the browser using alasql.js. The resulting table from the query is stored in local memory to be used later. After the user submits their own query, their SQL is again run using alasql.js. To check for correctness, the saved table and the user's table is compared using a compiled statement in Javascript:

```
var c = alasql.compile('SELECT VALUE ? == ?');
```

Then, the correct output and the user output can be compared using:

```
var correct = c([user_output, correct_output]);
```

Assuming the outputs match and are in the correct order, `correct` has a value of true after the user submits their code, regardless of whether or not it is identical code to the example answer. After this value is set, the system tells the user if they are correct if this value is true, or wrong if false. The user may attempt any question as many times as they need, and can continue to submit SQL queries even after they get a question correct. Once the user wants to move on to a new question, they may select “Next” to generate a new question and answer using the above automatas.

4.4 TABLE GENERATION

To supply both example and user queries with data, the environment is populated with two tables: Cars, and Register. To save on storage, both tables are generated at runtime using the following Javascript:

```
alasql("CREATE TABLE Cars (VIN INT, color
    STRING, price INT, brand STRING)");
alasql("CREATE TABLE Register (VIN INT, state
    STRING, date INT)");
for (let i = 0; i < 5000; i++) {
    alasql("INSERT INTO Cars VALUES (?, ?, ?,
        \'brand\')", [i, colors[Math.floor(Math
        .random() * colors.length)], Math
        .floor(Math.random() * 2000) + 5000])
    alasql("INSERT INTO Register VALUES (?, ?,
        ?)", [i, states[Math.floor(Math.random()
        * states.length)], Math.floor(Math
        .random() * 100) + 1990])
}
```

5 RESULTS AND FURTHER WORK

The repository can be located at https://github.com/Jar3dC0nway/SQLilly. The current system supports 2 tables, Cars and Register. Both schemas are available under the “Schema” button which can be toggled at any time. Additional tables would add more variety to the questions asked, and would make the questions less repetitive. Allowing users to submit their own tables would further add variety to the system, and could allow users to practice with data they know about. Such tables could be added in two different ways. The first is to allow users to add tables using pure SQL. The system could then detect the table names, data types, and foreign keys as this information should be included with the CREATE TABLE statement. An alternative approach is to allow the users to specify the schema and range of values for each column, similar to how Cars and Register is created.

Figure 3: Schema Statements. Clicking the Schema button will show the schema of the two tables over the answer box.

Each question can ask a maximum of 2 WHERE conditions, 1 GROUP BY, and 1 HAVING condition. There is currently no way to toggle which conditions the user wants to practice, but this is feasible to implement given the modular nature of the automatas. Additionally, when the user gets a question wrong, the system does not help the user in any way other than telling them they are right or wrong. Informing the user on information, such as how many extra rows they have or rows they are missing, could be one way to help users from remaining stuck. Alternatively, the user could choose to reveal the correct answer after 3 attempts, or reveal part of the correct answer.

The only aggregates being used are minimum, maximum, average, and count, although more would likely be needed. Other types of queries, such as CREATE or INSERT, would need their own automatas in order to be implemented in this approach. Additionally, such queries would need to reconstruct each set of tables after running either example SQL or user SQL. As tables are generated at runtime, the same code used to construct the tables could be reused for reconstruction.

REFERENCES

[1] Nada Basit, Abdeltawab Hendawi, Joseph Chen, and Alexander Sun. 2019. *A Learning Platform for SQL Injection. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (SIGCSE '19). Association for Computing Machinery, New York, NY, USA, 184–190. https://doi.org/10.1145/3287324.3287490

[2] Colin Beardon and John P. Wilson. 2013. *Experiential learning a handbook for Education, training and coaching* London: Kogan Page. Retrieved from: https://ebookcentral-proquest-com.proxy1.library.virginia.edu/lib/uva/reader.action?docID=1318826&ppg=12#

[3] UVA. 2024. Mayo Center - Experiential Learning, UVA Darden School of Business. Retrieved from: https://www.darden.virginia.edu/mayo-center/experiential-learning

[4] W3Schools. 2024. W3Schools Online Web Tutorials. Retrieved from: https://www.w3schools.com/sql/sql_exercises.asp