Journal of Network Intelligence
Taiwan Ubiquitous Information
 ISSN 2414-8105 (Online)
Volume **5**, Number **4**, November **2020**

Combining Dynamic and Static Analysis for Automated Grading SQL Statements

Jinshui Wang[1,2,3], Yunpeng Zhao[1,2,3], Zhengyi Tang[1,2,3,*], Zhenchang Xing[4]

[1]School of Computer Science and Mathematics,
Fujian University of Technology, Fuzhou,350118, China
[2]Fujian Provincial Key Laboratory of Big Data Mining and Applications,
Fujian University of Technology, Fuzhou,350118, China
[3]Key Laboratory of Hunan Province for Mobile Business Intelligence,
Hunan University of Technology and Business, Changsha 410205, China
[4]Research School of Computer Science, Australian National University, ACT, 2601, Australia
*Corresponding author: tangzy@fjut.edu.cn

Received August 2020; revised October 2020

Abstract. *Learning and teaching Structured Query Language (SQL) is a challenge that is widely recognized within the computer science education community. Programming activities are considered to be the most important aspect of the learning process. Manually grading of SQL statements, however, is often regarded as a tedious and time-consuming ordeal. In this paper we propose an automated grading approach for SQL statements, which combines dynamic and static analysis. The proposed approach first identified the correct statements from the student SQL statements, and then grades the incorrect statements based on the similarity between the statements and the correct statements. Empirical evaluation of the proposal was carried out on a dataset consisting of 255 manually graded pairs of statements. Experimental results show that our approach is superior to three state-of-the-art grading approaches. Our study found that each grading approach has its particular advantages and disadvantages, and the combination of different grading approaches can help to improve the quality of grading results.*
Keywords: Automated assessment, Automated grading, Structured Query Language

1. **Introduction.** As the dominant language in database language, SQL is regarded as an important subject in the degree course of computer science and information technology. While its syntax is relatively simple, it is still remarkably difficult to learn SQL [1, 2]. Courses of teaching SQL usually include assessments that give students a set of problems and ask them to construct SQL statements as the solutions. Such assessments not only measure students' understanding and performance, but also help the educational systems and learners to understand students' level of knowledge and gaps [3, 4]. Grading SQL statement is usually done by manually checking whether student SQL statement matches the correct statement, or by checking whether the result of student SQL statement matches that of the correct statement on one or more fixed datasets [5]. Checking and grading SQL statements manually takes time and is vulnerable to error. Moreover, the problem will become more and more serious as the number of students

and evaluation increases. This raises the need for an automated grading approach that can award marks to students based on their submitted SQL statements.

A number of automated grading approaches have recently been proposed for evaluating the work of students on SQL exercises. According to the evaluation strategy adopted, there are two automated grading approaches: static analysis approach and dynamic analysis approach. By executing the student statement and the reference statement provided by tutors on one or more fixed datasets, and then checking whether their results match, the dynamic analysis approach can determine whether the student statement is correct. The dynamic analysis approach is widely used in automated program grading systems and online judge systems [6]. However, dynamic analysis approach often fails to accurately assess how well the student has done based on the pre-set criteria, especially if the result of the student statement does not match the expected result. On the one hand, some student statements may be severely penalized for a small error, such as an extra column in the select clause. On the other hand, some statements containing serious errors may have good marks because the results are consistent with the expectations, such as both the student statement and the reference statement provided by the instructor produce an empty result. What's more, this approach cannot work at all if the student statements cannot be executed.

Unlike the dynamic analysis approach, the static analysis approach evaluates student SQL statements without executing them. The static analysis approach evaluates the structure of SQL statements, rather than their functions. The scores obtained through dynamic analysis approach and static analysis approach can differ greatly for the same SQL statement. A statement, for example, produces the expected result but its structure does not meet the evaluator's requirements set by the static analysis approach. Static analysis approach has the ability to generate detailed and descriptive feedback when analyzing SQL statements, to help students figure out their own errors. However, static analysis approach also has several drawbacks. Since there are many correct solutions for a SQL programming exercise, tutors have to provide as many solutions as possible so that one of them can match the student statement. And as the complexity of SQL exercises increases, the provision of all possible solutions becomes harder [7].

In this paper we proposed a hybrid automated grading approach for SQL statements. This approach uses both static and dynamic analysis to grade SQL statements. Dynamic analysis focuses on the identification of the correct statements submitted by students based on a comparison of statement results. This will compensate for problems in static analysis where it is difficult to provide adequate reference statements manually. Static analysis, on the other hand, focuses on specific features of SQL statements that can be used to generate fine-grained scores. We use dynamic analysis only to identify and grade the correct answers from the statements submitted by students, rather than to grade all SQL statements. After that, we compute partial marks for the remaining student statements with respect to all correct statements and pick the best match, that is, the one that assigns the highest marks. The rest of the paper is organized as follows: Section 2 presents related work on automated assessment methodologies and tools that have been developed. Section 3 describes our hybrid approach for automated grading of SQL statements. The results of this experimental study are presented in Section 4. In Section 5, we discuss and analyse these results. Finally, conclusions are presented in Section 6.

2. **Related Work.**

To assist in the teaching and learning of SQL, many research and educational systems were developed. Research related to our work may be roughly categorized into two areas, one is to help learn SQL, the other is to automatically score and evaluate SQL. Relevant literatures from each of the two areas below are briefly reviewed.

The first type of related research focuses on helping students learn SQL, while the Intelligent Tutoring System (ITS) provides a promising direction. The ITS is a computer-based program that presents educational materials in a flexible and personalized manner similar to one-to-one tutoring. The need for ITS in the field of SQL has been discussed in detail [8]. They are not only conducive to the development of an effective learning process [3], but also help to strengthen students' knowledge building [9]. SQL-Tutor [10] is an ITS designed as a guided discovery learning environment to help students learn SQL programming. SQL-Tutor has a very

simple architecture, which consists of user interface, pedagogical module and student modeler. The pedagogical module is the core of SQL-Tutor, which selects the problems to be given to students and generates corresponding instructional actions according to the student model. Bhagat et al. [11] provides students with an intelligent problem-solving environment where they can attempt to solve SQL problems presented by the system and obtain qualitative feedback on their solutions. In [12], an integrated Exploratorium is developed for database course. The Exploratorium provides access to three advanced educational activities: annotated samples, self-assessment questions, and SQL labs, each serviced by an independent web-based tool.

The other type of related research focuses on automated grading of SQL statements. A variety of automated assessment systems have been developed over the years. Many researchers have discussed these systems from different perspectives [6, 13, 14, 15]. Most automated grading systems and online judge platforms (e.g. CodeChef [1], CodeForces [2] and Hackerrank [3]) used dynamic analysis approach for grading. AsseSQL [16] is a typical example of a SQL assessment system. AsseSQL grades the student's SQL statement by comparing the results of the execution of the reference statement with the results of executing the student statement. If the comparison results are the same, the student's statement is marked as correct, otherwise it is marked as incorrect. Similar to AsseSQL, most automated grading systems (e.g. SQLator [1], Aplicación BD [17], SQL-ACME [18], DB-Learn [19]) involve executing student's SQL statement and reference statements on a dataset and grading them based on whether the execution results are the same. These systems gave much more immediate and informative feedback than that provided by paper-based examination. However, they only produce a binary grade (correct or incorrect). Even if student's statement has only a few errors, it will be judged as completely incorrect because its execution result is different from that of the reference statement.

To improve students' learning experience and the efficiency of evaluation, a comprehensive teaching and evaluation tool for SQL writing skills called SQLify was proposed [20]. Instead of simply providing correct or incorrect results, SQLify introduced peer review to address continuous grading issues, which produced a wider grading range. Kleiner et al. [21] presented a tool called aSQLg, which adopted a multi-step marking algorithm and taken several factors (i.e. syntactical correctness, efficiency of statement, correctness of results and style) of student's statement into consideration. Apart from these factors, aSQLg also allowed an additional manual grading step by an instructor. Bhangdiya et al. [22] presented the XDa-TA system for automated grading of SQL statements. Given one or more correct statements for an SQL assignment, the tool automatically generated datasets dedicated to capturing common errors. The grading was then done by comparing the execution results of students' statement with those of the correct statements against these generated datasets. Kleerekoper and Schofield [2] presented an online assessment tool called SQL Tester which was very similar to AsseSQL and assessed its impact on student engagement and performance. The results from a student questionnaire showed that 90% of the students wanted to spend more time using the tool to get good marks, and 75% agreed that the tool motivated them to revise.

In the last years, several methodologies have been applied for grading SQL statements in a continuous scale [0%, 100%]. Chandra et al. [23] extended the XData data generation techniques [22, 24, 25] to cope with more diversified SQL statements and a larger class of mutations, and built a system for grading SQL using the datasets generated by XData. They evaluated their grading tool and showed that it was better at catching student query errors than fixed datasets or correction by teaching assistants. Stajduhar and Mausa [26] proposed an automated method for grading individual SQL statements. The proposal used several common and simple string similarity metrics for comparing the student's statement with the reference statements. These data were then used to construct the predictive logistic regression model together with the manually assigned grades. The model achieved the expected classification accuracy of 78% in binary classification, which showed its potential in practical application. Chandra et al. [27]

[1]https://www.codechef.com/
[2]https://www.codeforces.com/
[3]https://www.hackerrank.com/

extended the XData system by adding features that enable it to grade partially correct statements. They first checked for correctness by using datasets generated by the XData system. If a statement is marked as incorrect, then several techniques were used to award partial marks. Chu et al. [28] introduced COSETTE, a prover that can determine whether two SQL queries are semantically equivalent, and demonstrated an automated grading result of a SQL homework problem using COSETTE. Although these systems are capable of partial grading, they require tutors to provide statements that can deduce other correct statements through different ways such as equivalence calculation or mutation operation.

3. Methodology.

In this section, we first described how the student SQL statements were partitioned. After that, the formula used to measure the similarity of SQL statements was introduced. Finally, we present the hybrid approach used to grade SQL statements automatically.

3.1. Statement partition.

There are usually many different correct statements for a SQL exercise, and only one of them is provided as the reference statement by tutors. Considering that there may be large differences between these different correct statements, the student statements should not be graded solely based on their textual or syntactic similarities with a single reference statement. It is, therefore, necessary to identify as many different correct statements as possible, and then grade students' SQL statements based on the similarity between students' SQL statements and each correct statement. Besides, numerous automated grading approaches [5, 29, 30] have been proposed based on the syntax structure of SQL statements. Compared to text-based approaches, these approaches focused on the syntax structure are not influenced by variations in aliases and the order of expression.

However, these methods would no longer be valid for those statements with syntax errors, as the statements cannot be parsed. Therefore, it is necessary to classify SQL statements according to whether they can be parsed or not, to pick specific grading strategies for them.

For a given problem and a database instance, the execution results of all correct SQL statements must be the same. For convenience, we denote the reference statement by rs and the set of student's statements by SS. According to the execution result of the statement, the following exclusive rules were used to divide SS into three sets, namely $SS = CS \cup NS \cup PCS$, where:

- CS is the set of correct statements, it is defined as $CS = \{s \mid exec(s) = exec(rs)\}$, where the function $exec(s)$ means the result of statement s on the given database instance.
- NS is the set of non-executable statements, it is defined as $NS = \{s \mid exec(s) = Fail\}$, where $Fail$ means the statement cannot be interpreted, i.e. DBMS reports an error.
- PCS is the set of partially correct statements, it is defined as $PCS = \{s \mid exec(s) \neq exec(rs) \wedge exec(s) \neq Fail\}$.

3.2. Similarity calculation.

For statements in PCS, their syntax information can be represented by abstract syntax trees (ASTs). In computer science, AST is a tree representation of the abstract syntax structure of source code written in a programming language. Fig. 1 illustrates the AST of an example SQL statement Q_1. Compared with source code, the AST representation does not contain elements that are not important for calculating similarities, such as comments, semicolons, and parentheses, but also can reduce the potential adverse effects of aliases. Therefore, AST is widely used to analyze the semantic structure of SQL statements.

Q_1:
```
Select p.ID, p.first_name
From person p Join writer w On p.id = w.id
Where Lower(p.last_name) = 'Allen';
```

Note that the AST is an ordered tree, but the order of nodes in some expressions or clauses does not affect the function of the SQL statement. For Q_1, the expression "*FROM Person p join Writer w on p.id = w.id*" and the expression "*FROM Writer w join Person p on p.id = w.id*" are functionally equivalent, but their corresponding ASTs are different. Therefore, those order- independent nodes will be identified and then sorted alphabetically before the similarity is calculated. In addition, SQL keywords are not case-sensitive, so lower-case letters are equivalent to corresponding upper-case letters except within string and character literals.

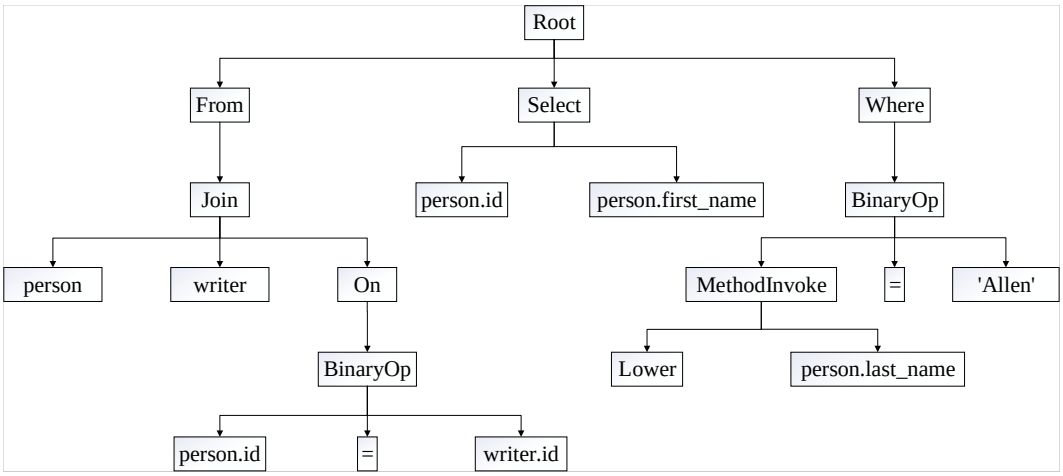

FIGURE 1. The abstract syntax tree of Q_1.

Therefore, all letters except those within the string and character literals are converted to corresponding lower-case letters. Finally, if there was no sorting requirement for the execution results, the node of "order" in AST is not considered when calculating the similarity.

There are three different algorithms for measuring tree similarity [34]: top-down maximum common subtree isomorphism, bottom-up maximum common subtree isomorphism, and tree edit distance (TED). Compared with the first two algorithms, the similarity of two ASTs can be calculated more accurately by using TED [35]. The TED between two trees is determined as the minimum-cost sequence of node edit operations that transform one tree into another. Various algorithms have been proposed and evaluated to calculate TED [36, 37, 38]. In this paper, we apply a robust and memory-efficient algorithm namely APTED [37] to computer the TED between ASTs. We use $APTED(s_1, s_2)$ to denote the edit distance between two ASTs s_1 and s_2. Let $|T|$ denotes the total number of nodes in AST T, the similarity of two ASTs s_1 and s_2 then is the $APTED(s_1, s_2)$ normalized using the method described in [39].

$$ASTSim(s_1, s_2) = 1 - \frac{2 \times APTED(s_1, s_2)}{|s_1| + |s_2| + APTED(s_1, s_2)} \tag{1}$$

Since ASTs cannot be built for SQL statements in NS, their textual features were used to calculate similarity. In each SQL statement string, both in the student's statement and in the reference statement, all newline, tab, and semicolon characters were replaced with whitespace. In addition, all leading and trailing whitespaces were removed, and all adjacent intermediary whitespaces were combined into one single whitespace. The Normalized Levenshtein distance metric was then used to calculate the similarity of two statements. The Levenshtein distance $LevDist(s_1, s_2)$ is the minimum number of single-character edits (i.e. insertions, deletions, or substitutions) required to transform one statement into the other. The similarity of s_1, s_2 is calculated by the following equation:

$$TxtSim(s_1, s_2) = 1 - \frac{LevDist(s_1, s_2)}{max(length(s_1), length(s_2))} \tag{2}$$

3.3. **Statement grading.** By referring to the grading strategy of the dynamic analysis approach, all statements in the CS are regarded as correct solutions. All statements in CS can, therefore, be regarded as the grading criteria for SQL statements in NS and PCS. Grading based on the similarity with multiple statements in CS, rather than a few given reference statements, helps to broaden the set of grading criteria, thereby alleviating the problem that the static analysis approach relies too much on the quality and quantity of given reference statements. When there was more than one correct statement to an SQL exercise, the correct

statement which was the most similar to the student's statement was used as the criterion for calculating the grade.

Here, the pair of a statement s and its grade g is a tuple that can be labelled as $gs = (s, g)$, and the set of pair of gs is labelled as GS. The pseudocode of the hybrid grading algorithm to grade statement is given in Algorithm 1.

Algorithm 1: The hybrid approach for automated grading SQL statement.

Input: The set of students' statements: SS; reference statement: rs
Output: The set of pair of statement and its grade: GS

$CS = \{rs\}$;
$NS = \emptyset$;
$PCS = \emptyset$;
foreach $s \in SS$ **do**
  if $exec(s) = exec(rs)$ **then** $CS = CS \cup \{s\}$;
  else if $exec(s) = Fail$ **then** $NS = NS \cup \{s\}$;
  else if $exec(s) \neq exec(rs)$ **then** $PCS = PCS \cup \{s\}$;
end
$GS = \emptyset$;
foreach $s \in CS$ **do**
  $GS = GS \cup \{(s, 100\%)\}$;
end
foreach $s \in NS$ **do**
  $g = max(\{TxtSim(s, s') \mid s' \in CS\})$;
  $GS = GS \cup \{(s, g)\}$;
end
foreach $s \in PCS$ **do**
  $g = max(\{ASTSim(s, s') \mid s' \in CS\})$;
  $GS = GS \cup \{(s, g)\}$;
end
return GS ;

4. **Experimental Evaluation.** Our experimental evaluation has two main objectives: (1) to verify whether the proposed hybrid approach could be successfully used for automated grading of SQL statements, and (2) to identify particular strengths and weaknesses of different grading approaches. To evaluate the proposed hybrid grading approach, we compared the hybrid approach HA with the following different approaches.

Dynamic analysis (DA): executes the student statement and compares the results with the expected one. If the result of a statement is the same as that of the reference statement, then the statement is graded 100%, otherwise 0%.

Syntax-based analysis (SA): calculates the syntax similarity between the student statement and the reference statement by equation 1, and then takes the similarity as the grade of the student SQL statement. Set the grade to 0% for the statement that cannot build an AST.

Text-based analysis (TA): calculates the textual similarity between the student statement and the reference statement by equation 2, and then takes the similarity as the grade of the student SQL statement.

4.1. **Data collection.** Our experiment was conducted in an undergraduate Relational Database course at the Australian National University. The experiment was conducted on August 10th 2018 when students enrolled in the Relational Database course started to learn relational data model and SQL. The experiment was carried out fully online for three weeks and a total of 393 students were enrolled. The students were asked to login in an online assessment platform

and complete 15 exercises. This platform provided an SQLite environment in students' browsers by compiling the SQLite C code with Emscripten. Students were allowed to submit and execute their answers in the form of SQL statements. If the execution result of the statement submitted by the student is the same as that of the reference statement, the online assessment platforms will return a feedback message indicating that the execution result is correct. During the interaction with the assessment platform, statements submitted by students were recorded and archived.

Overall, our experiment had collected 12,899 statements submitted by students. To create a benchmark dataset that can be used to evaluate different grading approaches, we randomly selected 45 SQL statements submitted by students for each exercise, and asked three teaching assistants to grade them manually. Finally, we average the scores provided by the three assistants and take it as the final score of each statement. The dataset collected in this experiment is ready for public release [4].

4.2. **Performance measures.** To compare different grading approaches, an accuracy metric is usually defined in terms of the forecasting error which is the difference between the actual value (calculated by teacher assistances) and the predicted value (calculated by grading approaches). There are a number of metrics of accuracy in the related literatures are:

- **Mean Absolute Error (MAE)** measures the average magnitude of errors in a set of predictions without considering their direction. It is the average value of the absolute difference between the predicted value and the actual observation value on the test sample, in which all individual differences have equal weight.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |A_t - P_i|}{n} \tag{3}$$

- **Symmetric Mean Absolute Percentage Error (SMAPE)** is an accuracy measure based on percentage errors. SMAPE is a modified Mean Absolute Percentage Error (MAPE) in which the divisor is half of the sum of the actual and forecast values.

$$\mathrm{SMAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \frac{|F_t - A_t|}{(|A_t| + |F_t|)/2} \tag{4}$$

- **Root Mean Square Error (RMSE)** is a quadratic scoring rule, which can also measure the average magnitude of the error. It is the square root of the average value of squared differences between the predicted value and the actual value.

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} (A_i - P_i)^2}{n}} \tag{5}$$

where A and P stands for the actual and predicted value, respectively, and n stands for the size of sample. Both MAE and RMSE range from 0 to ∞, and are indifferent to the direction of errors. In addition, all three metrics are negatively oriented scores, that is, lower values are better.

4.3. **Results.** Descriptive statistics of the benchmark as well as the grading summary of different approaches are given in Table 1. In terms of the median and mean values, the grades calculated by the HA are generally higher than those by other approaches, mainly due to the following aspects. First of all, the DA scored 0% for SQL statements whose execution results differ from the expected ones, while the HA would award partial marks based on the similarity between those statements and correct statements. Second, both SA and TA graded statements based on the similarity between the student's SQL statement and a given reference statement, while the HA first identified the correct statement that is most similar to the student's statement, and then graded it based on their similarities.

As shown in Table 1, the maximum values calculated by the four grading approaches are the same. When the maximum value calculated by TA is 100%,

[4]https://gitee.com/submitpaper/Automated-Grading-of-SQL-Statements

TABLE 1. Descriptive statistics of benchmark as well as grading summary of different approaches.

	Benchmark (%)	SA (%)	TA (%)	DA (%)	HA (%)
Mean	79.04	55.04	51.38	59.11	86.44
Median	96.67	51.28	45.28	100	100
Min	1.67	0	3.99	0	18.34
Max	100	100	100	100	100
Standard deviation	28.35	31.48	25.23	49.27	21.33

it means that there is at least one statement submitted by student identical to a reference statement. The syntax structure and execution result of this statement and the reference statement must be the same, resulting in a full score calculated by HA, SA, and DA. Different from the maximum value, the minimum values calculated by the four approaches are different. According to the grading strategy of the SA, there must be at least one SQL statement that cannot construct an AST for it, which will cause its score to be 0%. Correspondingly, the statement must also have syntax errors and cannot be parsed, resulting in DA also scored it as 0%. As expected, the minimum value of the grading results by HA is much higher than that of other approaches. First, according to the minimum value of the results graded by TA (3.99%), for all the SQL statements submitted by the students, there will be at least a certain degree of similarity in the text with the reference statement, that is, there is no blank SQL statement. Second, for SQL statements that cannot be parsed, HA calculates their scores based on the textual similarity between them and the reference statements. Finally, regardless of whether students' statements can be parsed or not, HA takes the maximum similarity between students' statements and all correct statements as their scores, so the minimum value of Hybrid grading result is likely to be higher than that of other approaches. Table 2 summarizes the performance of the four grading approaches. The comparison results showed that HA has the smallest MAE, SMAPE, and RMSE, indicating that HA has a clear advantage over other approaches. Furthermore, it is interesting that none of the SA, TA, and DA is significantly better than the other two.

TABLE 2. Performance of different grading approaches.

Approach	MAE	SMAPE (%)	RMSE
SA	26.01	54.87	34.20
TA	29.89	50.25	37.15
DA	21.66	82.73	36.18
HA	**8.37**	**17.81**	**14.67**

5. **Discussion.** The experimental results showed that the existing grading approach (i.e. SA, TA, and DA) could not yield satisfactory results. On one hand, while the dynamic analysis approach can effectively determine if the statements are correct by comparing the execution results of SQL statements, it cannot award partial marks for those statements that are partially correct or cannot be executed. Therefore, this led to a high standard deviation (49.27%) in the

grading results of dynamic analysis. On the other hand, all statements can be graded accordingly based on the similarity between the student's statements and the reference statements. However, whether the reference statements can fully cover the features of the various correct statements will have a significant impact on the accuracy of the grading results. If only one reference SQL statement is provided for each exercise, the grading results of static analysis are likely to be unsatisfactory.

In contrast, the proposed hybrid approach first identifies various correct statements submitted by students, and then grades other statements according to their similarities with all the identified correct statements. Benefiting from the identified correct SQL statements, the SQL statements submitted by students have the opportunity to compare with the most similar statements rather than confining themselves to a few given reference statements. Therefore, the hybrid approach has the ability to evaluate statements more comprehensively and accurately, and its grading results are closest to the results of manual grading (details in Table 2). It also means that there are some deficiencies in the current practices [6, 37] of using online judge systems to grade programming assignments. The online judge system computes an aggregated score for the program based on the results of all considered test cases. When the SQL statement fails to pass the test cases due to some minor errors, the online judge systems will award an unreasonable low score.

6. **Conclusions.** In this paper we proposed a hybrid approach for automated grading of SQL statements, and compared it with three state-of-the-art approaches (i.e. dynamic analysis approach, syntax-based analysis approach, and text-based analysis approach). To the best of our knowledge, this is the first study that quantitatively compares different automated grading approaches. Experimental results show that these three state-of-the-art approaches have their shortcomings, which leads to a large difference between their grading results and manually grading results. Due to the proposed hybrid approach combines dynamic analysis and static analysis to take advantage of the complementary features of each and overcome their shortcomings, the experimental results also show that the hybrid approach is superior to other approaches.

As a grading approach, the proposed hybrid approach only awards marks for students SQL statements, and does not provide detailed feedback on errors in statements. In the future work, we will provide meaningful textual feedback according to the differences between students statements and correct statements to help students improve their solutions. This will meet the formative way of evaluation that consists of diagnosing students' weaknesses and guiding them in the right direction [41]. Moreover, test data generation and query equivalence will be added to the next version of the proposed hybrid approach to further improve the accuracy of grading results.

Acknowledgment. This work was supported by Fujian Provincial Education Science "Thirteenth Five-Year Plan" Project under Award FJJKCG19-001, Education and Research Projects of Fujian University of Technology under Award JXKA18015 and Award GY-Z15101, and the Open Fund of Key Laboratory of

Hunan Province for Mobile Business Intelligence of Hunan University of Technology and Business (2015TP1002).

REFERENCES

[1] S. Sadiq, M. Orlowska, W. Sadiq, and J. Lin, Sqlator: an online sql learning workbench, *Acm Sigcse Bulletin*, vol.36, no.3, pp.223–227, 2004.

[2] A. Kleerekoper and A. Schofield, Sql tester: an online sql assessment tool and its impact, *Proc. of the the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*,Larnaca, Cyprus, pp.87–92, 2018.

[3] F. Grivokostopoulou, I. Perikos, and I. Hatzilygeroudis, An educational system for learning search algorithms and automatically assessing student performance, *International Journal of Artificial Intelligence in Education*, vol.27, no.1, pp.207–240, 2017.

[4] Z. Jeremic, J. Jovanovic, and D. Gasevic, Student modeling and assessment in intelligent tutoring of software patterns, *Expert Systems with Applications*, vol.39, no.1, pp.210—222, 2012.

[5] B. Chandra, A. Banerjee, U. Hazra, M. Joseph, and S. Sudarshan, Automated grading of sql queries, *Proc. of the IEEE 35th International Conference on Data Engineering (ICDE)*, Macao, Macao, pp.1630—1633, 2019.

[6] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, A Survey on Online Judge Systems and Their Applications, *ACM Computing Surveys*, vol.51, no.1, pp.1–34, 2018.

[7] M. Vujošević-Janičić, M. Nikolić, D. Tošić, and V. Kuncak, Software verification and graph similarity for automated evaluation of students' assignments, *Information & Software Technology*, vol.55, no.6, pp.1004—1016, 2013.

[8] A. Mitrovic, A knowledge-based teaching system for sql, *Proc. of the ED-MEDIA/ED-TELECOM 98 Conference*, Freiburg, Germany, pp.1027—1032, 1998.

[9] N. T. Heffernan and C. L. Heffernan, The assistments ecosystem: Building a platform that brings scientists and teachers together for minimally invasive research on human learning and teaching, *International Journal of Artificial Intelligence in Education*, vol.24, no.4, pp.470—497, 2014.

[10] A. Mitrovic, Learning sql with a computerized tutor, *Proc. of the 29th SIGCSE technical symposium on Computer science education*, Atlanta Georgia, USA, pp.307—311, 1998.

[11] S. Bhagat, L. Bhagat, J. Kavalan, and M. Sasikumar, Acharya: An intelligent tutoring environment for learning sql, *Proc. of the Vidyakash 2002 International Conference on Online Learning*, Mumbai, India, pp.15—17, 2002.

[12] 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. *ACM Transactions on Computing Education (TOCE)*, vol.9, no.4, pp.1—15, 2010.

[13] R. Queirós and J. P. Leal, Programming exercises evaluation systems-an interoperability survey. *Proc. of the 4th International Conference on Computer Supported Education*, vol.2, Porto, Portugal, pp.83—90. 2012.

[14] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, Review of recent systems for automatic assessment of programming assignments, *Proc. of the 10th Koli calling international conference on computing education research*, Koli Finland, pp.86—93, 2010.

[15] K. M. Ala-Mutka, A survey of automated assessment approaches for programming assignments, *Computer science education*, vol.15, no.2, pp.83—102, 2005.

[16] J. C. Prior and R. Lister, The backwash effect on sql skills grading, *ACM SIGCSE Bulletin*, vol.36, no.3, pp.32—36, 2004.

[17] C. Domínguez, A. Jaime, J. Heras, and F. J. García-Izquierdo, The effects of adding non-compulsory exercises to an online learning tool on student performance and code copying, *ACM Transactions on Computing Education (TOCE)*, vol.19, no.3, pp.1—22, 2019.

[18] J. Soler, F. Prados, I. Boada, and J. Poch, A web-based tool for teaching and learning sql, *Proc. of the 7th International Conference on Information Technology Based Higher Education and Training*, Ultimo, Australia, 2006.

[19] S. Nalintippayawong, K. Atchariyachanvanich, and T. Julavanich, Dblearn: Adaptive e-learning for practical database course—an integrated architecture approach, *Proc. of the 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Kanazawa, Japan, pp.109—114, 2017.

[20] S. Dekeyser, M. D. Raadt, and T. Y. Lee, Computer assisted assessment of sql query skills, *Proc. of the 18th conference on Australasian database*, vol.63, Victoria, Australia, pp.53—62, 2007.

[21] C. Kleiner, C. Tebbe, and F. Heine, Automated grading and tutoring of sql statements to improve student learning, *Proc. of the 13th Koli Calling International Conference on Computing Education Research*, Joensuu, Finland, pp.161—168, 2013.

[22] A. Bhangdiya, B. Chandra, B. Kar, B. Radhakrishnan, K. M. Reddy, S. Shah, and S. Sudarshan, The xda-ta sys- tem for automated grading of sql query assignments, *Proc. of the IEEE 31st International Conference on Data Engineering*, Seoul, South Korea, pp.1468–1471, 2015.

[23] B. Chandra, B. Chawda, B. Kar, K. V. Reddy, S. Shah, and S. Sudarshan, Data generation for testing and grading sql queries. *The International Journal on Very Large Data Bases*, vol.24, no.6, pp.731—755, 2015.

[24] B. Chandra, B. Chawda, S. Shah, S. Sudarshan, and A. Shah, Extending xdata to kill sql query mutants in the wild, *Proc. of the 6th International Workshop on Testing Database Systems*, New York, USA, pp.1—6, 2013.

[25] S. Shah, S. Sudarshan, S. Kajbaje, S. Patidar, B. P. Gupta, and D. Vira, Generating test data for killing sql mutants: A constraint-based approach, *Proc. of the IEEE 27th International Conference on Data Engineering*, Hannover, Germany, pp.1175—1186, 2011.

[26] I. Stajduhar, and G. Mausa, Using string similarity metrics for automated grading of sql statements, *Proc. of the 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, pp.1250—1255, 2015.

[27] B. Chandra, M. Joseph, B. Radhakrishnan, S. Acharya, and S. Sudarshan, Partial marking for automated grading of sql queries, *Very Large Data Bases*, vol.9, no.13, pp.1541—1544, 2016.

[28] S. M. Chu, D. Li, C. L. Wang, A. Cheung, and D. Suciu, Demonstration of the cosette automated sql prover, *Proc. of the 2017 ACM International Conference on Management of Data*, Chicago Illinois USA, pp.1591—1594, 2017.

[29] P. Guagliardo and L. Libkin, A formal semantics of sql queries, its validation, and applications, *Very Large Data Bases*, vol.11, no.1, pp.27—39, 2017.

[30] D. Insa and J. Silva, Automatic assessment of java code, *Computer Languages, Systems & Structures*, vol.53, pp.59—72, 2018.

[31] H. Ma, T.-Y. Wu, M. Chen, R.-H. Yang, J.-S. Pan, A Parse Tree- Based NoSQL Injection Attacks Detection Mechanism, *Journal of Information Hiding and Multimedia Signal Processing*, vol.8, no.4, pp.916-9-28, 2017.

[32] T.-Y. Wu, C.-M. Chen, X. Sun, C.W. Lin, A countermeasure to SQL injection attack for cloud environment, *Wireless Personal Communications*, Vol.96, no.4, pp 5279—5293, 2017.

[33] T.-Y. Wu, J.S. Pan, C.M. Chen, C.W. Lin, Towards SQL injection attacks detection mechanism using parse tree, *Proc. of the 8th International Conference on Genetic and Evolutionary Computing (ICGEC 2014)*, Nanchang, China, pp. 371–380, 2015.

[34] G. Valiente, Algorithms on Trees and Graphs, *Springer Science & Business Media*, 2013.

[35] T. Sager, A. Bernstein, M. Pinzger, and C. Kiefer, Detecting similar java classes using tree algorithms, *Proc. of the 2006 International Workshop on Mining Software Repositories*, Shanghai, China, pp.65—71, 2006.

[36] M. Pawlik and N. Augsten, Rted: a robust algorithm for the tree edit distance, *Very Large Data Bases*, vol.5, no.4, pp.334—345, 2011.

[37] M. Pawlik and N. Augsten, Tree edit distance: Robust and memory-efficient, *Information Systems*, vol.56, pp.157–173, 2015.

[38] R. Sridharamurthy, T. B. Masood, A. Kamakshi-dasan, and V. Natarajan, Edit distance between merge trees, *IEEE Transactions on Visualization and Computer Graphics*, vol.26, no.3, pp.1518—1531, 2020.

[39] L. I. Yujian and C. G. Zhang, A metric normalization of tree edit distance, *Frontiers of Computer Science in China*, vol.5, no.1, pp.119—125, 2011.

[40] W. J. Zhou, Y. G Pan, Y. H. Zhou, and G. Z. Sun, The framework of a new online judge system for programming education, *Proc. of the ACM Turing Celebration Conference-China*, Shanghai China, pp.9—14, 2018.

[41] S. M. Arifi, I. N. Abdellah, A. Zahi, and R. Benabbou, Automatic program assessment using static and dynamic analysis, *Proc. of the 3rd World Conference on Complex Systems (WCCS)*, Marrakech, Morocco, pp.1—6, 2015.