Sensitivity of Automated SQL Grading in Computer Science Courses

Benard Wanjiru(✉), Patrick van Bommel, and Djoerd Hiemstra

Radboud University, Nijmegen, The Netherlands
{benard.wanjiru,patrick.vanbommel,djoerd.hiemstra}@ru.nl

Abstract. Previous research has primarily relied on fixed procedures when implementing partial grading systems. As a result, the sensitivity of such systems in terms of error analysis becomes inflexible as well. In this paper, we employ a software correctness model that allows for a dynamic and flexible approach for adjusting the sensitivity of a grading system based on the user's needs and goals. We show how partial grading can be used to award fair grades and also categorize students into groups based on their strengths and weaknesses observed in their answers. Furthermore, we show how the sensitivity of a grading system can be varied to allow such grouping. To illustrate this, we analysed more than 2000 answers for 6 SQL programming assignments. An implication of this study is that instructors can carry out more effective partial grading of SQL queries as well as adjust learning material based on the needs of a particular group of students. They can address the observed limitations, thereby bridging the gap between high-performing students and those that require additional attention.

Keywords: correctness levels · software correctness · automated grading · assessment · partial marks · SQL query grading

1 Introduction

Manually grading software exercises can take a lot of time and become a tedious task for instructors, particularly when dealing with large numbers of submissions. Furthermore, it can lead to fatigue and errors, making automated grading systems a worthy alternative. These grading systems fall into two categories: binary grading systems and partial grading systems. In respect to SQL, binary grading is the lowest form of grading whereby the query is either correct or incorrect. One way of implementing this is executing the query and checking its results [12]. Binary grading can be demotivating for students. Specifically, if a simple syntax mistake leads to a zero score. Consider a student who clearly understands a topic. They put effort into a given exercise but end up making a simple mistake like misspelling a keyword. Misspells might be due to some circumstances for example: having too little time or being inexperienced with the development tools used for the assignment. In this case, receiving a zero or

K. Daimi and A. Al Sadoon (Eds.): ICR 2024, LNNS 1058, pp. 283–299, 2024.
https://doi.org/10.1007/978-3-031-65522-7_26

fail might leave them feeling unfairly graded and demotivated. For this reason, partial grading becomes important. In this form of grading, points are awarded for correct parts of the query, and constructive feedback is provided for the parts that need improvement. Partial grading systems make it possible to acknowledge students' good work and point out where they need to improve. This approach gives the student a sense of accomplishment therefore protecting their motivation and enhancing it as well.

Previous research has extensively explored binary grading systems. For instance, a tool capable of generating various data-sets for evaluating students' solutions [2]. This tool carries out assessment by comparing students' solutions with the correct solution provided by the instructor. The limitation of binary grading systems is that they only check query results. Additionally, partial grading systems have been explored as well. For example, SQLify [6], is able to assess students query skills using 8 levels. The tool focuses on giving students a chance to learn from each other by offering peer review and semantic feedback. Similarly, aSQLg [9] offers automatic assessment of SQL statements as well as an environment for students to practice and perfect SQL skills by relying on the systems feedback. This tool combines checks for syntax, result and style with manual grading. Moreover, tools capable of awarding partial marks to incorrect queries using edit distances between the correct query and incorrect query have been proposed [3].

However, much research up to now has primarily relied on fixed procedures when implementing partial grading systems. These systems don't allow for the flexibility of giving more weight to specific properties like syntax or results based on the needs of the instructor. Furthermore, they do not offer the flexibility of adding or removing properties during grading. For instance, some research uses syntax, schema, results and peer review in their assessment [6]. Introducing another property, for example, style, into this assessment would require a major alteration to this method. Let us consider an instructor who wants to emphasize syntax assessment in an introductory topic of a computer science class. In this scenario, the grading would weigh syntax more heavily than other properties. As the course progresses, the instructor might turn their attention to semantics. More weight when grading would be given to semantics as opposed to syntax. Eventually, they might shift the focus to results particularly for critical software. In this scenario, the software must not fail, therefore, more weight is emphasized on the results. The problem with previous methods, as shown here, is that they are not flexible enough to allow for dynamic weighing of properties dependent on the instructor's goals.

To address these limitations, we propose a flexible approach using a software correctness model. This model allows the flexibility of adding or removing properties at will therefore, allowing us to adjust the sensitivity of an automated grading system depending on the goals of assessment. Furthermore, the model allows giving more weight to a specific property during grading. This enables highlighting a specific property like syntax, that the instructor thinks is more important to evaluate at a certain stage of the teaching process. These two fea-

tures allow us to better provide partial grades based on the objectives of the instructor. Also, we use this model to categorize students into groups based on their strengths and weaknesses observed in their answers. This grouping not only highlights those areas where students do well, but also identifies where they require additional attention. Consequently, instructors are then able to recognize what specific students need in terms of teaching materials.

Towards this goal, we graded 2113 students SQL queries using this software correctness model instantiated using various parameters. The subsequent analysis of the results seen informed our conclusions.

2 Background

2.1 Software Correctness in Software Exercises

Software correctness in software exercises is the degree to which code written by students satisfies given specifications and meets instructor's expectations [14]. Software correctness can be assessed through testing techniques. The goal of testing is to make sure that students are able to write software that is reliable, predictable, and free of errors. For instance, in languages like C++, software correctness is often tested using techniques like unit testing [16]. However, the binary testing nature of unit testing makes it less effective for SQL partial grading. Nevertheless, there are other metrics that can be used to measure software correctness in SQL. For example, executing the query and checking the degree of variation between the output and the expected results.

2.2 Software Correctness Levels

Software correctness levels describe the degree to which code written by students adheres to an instructor's specifications. These levels are defined as discrete values ranging from "fully correct" to "completely incorrect." Automated grading tools that feature at least two levels of correctness, correct and incorrect have already been proposed [2,10]. These are binary grading systems, characterized by categorizing an answer as either correct or incorrect.

Binary grading systems have numerous limitations. They lack granularity in differentiating between errors. This often leads to perceived unfairness among students, as submissions with minor errors are judged as fully incorrect. Furthermore, they make it difficult to provide feedback as specific errors cannot be pinpointed. Our model offers more possible outcomes to enable differentiation of errors.

Due to binary grading limitations, tools that incorporate partial grading of software exercises have been proposed [3,5,7,8]. Correctness levels enable the development of partial grading systems. The greater the number of correctness levels a tool possess, the more insight it is has. For example, utilizing 8 correctness levels to differentiate syntax, output schema, and semantics errors as seen in SQLify [6]. SQLify is able to automatically assign 5 levels, the remaining 3 levels

are assigned manually using peer review. Furthermore, each property checked by the system is assigned two outcomes: correct or incorrect. Our model is fully automated, assigning more than 8 levels. Moreover, we are able to assign more than two outcomes to each property checked.

2.3 Properties

Properties refer to distinctive features of code that can be examined to evaluate its quality. These features may include syntax, semantics, and other relevant factors that can be evaluated using a program. These properties contribute to the overall functionality of the whole software. Understanding what properties of code are, their effects on the overall software and how they can be evaluated is essential. It helps in determining the appropriate levels of correctness required for automated grading systems to evaluate software exercises effectively.

There exist numerous properties that can be employed to assess code quality. For instance, Style++ examines the style of C++ code produced by students [1]. Similarly, tools capable of assessing complexity in C++, have been proposed [15,18]. For more sophisticated applications that employ multi-threading, performance becomes a critical property to measure [13].

The choice of properties to check in a grading system is dependent on the learning objectives of the course and the constraints imposed by the programming language used for creating the submissions. Some of these properties are: syntax, semantics, results, complexity and efficiency. This list does not encompass all the possible properties, additional properties can be added or removed as necessary.

Syntax is the first property to check since it is the foundation for creating executable software. Semantics, on the other hand, is used to check whether written code can perform a given task. Results are useful only in cases where there is an output. It is also possible to write code using various code writing approaches, some of which may contain redundant or unnecessary elements. In such cases, assessing complexity may be necessary. Efficiency may be an important property to check in some cases. Specifically, when the code is used in resource constrained environments such as embedded systems, mobile devices, and IoT devices with limited processing power, memory, and battery life. Efficient software can operate in these environments reliably by consuming fewer resources, including execution time, memory, and network bandwidth.

2.4 Outcomes

When evaluating the properties discussed above, it is crucial to assign a measure of quality to each property from a list of categories. We refer to these categories as outcomes of the properties. For instance, some studies use two outcomes: correct and incorrect [6]. However, the number of outcomes used may vary based on the sensitivity of a grading system and its use case. A more sensitive grading system would require a higher number of outcomes to ensure that grading is accurate. Some of example outcomes of the properties listed in Subsect. 2.3 are:

- Correct: The code meets full specifications.
- Minor Incorrect: The code has minor errors that can be discarded.
- Major Incorrect: The code has some errors that do not impact the overall functionality.
- Absolutely Incorrect: The code has fatal errors.

2.5 The Generic Model for Defining Software Correctness Levels

In this subsection, we introduce the generic model for defining correctness levels needed by a grading system. As discussed in Subsect. 2.3, various properties can be used to evaluate quality of code. In the generic model, we will use a set P to hold the necessary properties. Similarly, as mentioned in Sect. 2.4, multiple outcomes can be associated with each property. In the generic model, we will use a set T to hold all relevant outcomes. The possible outcomes for a specific property is now given by the following function:

$$Outcomes : P \rightarrow \mathbb{P}(T) \tag{1}$$

where $\mathbb{P}(T)$ is the powerset of T. For $x \in P$, we have $Outcomes(x)$ giving possible outcomes of property x.

When specific properties with their outcomes have been chosen, the correctness levels can be generated. The number of correctness levels ℓ is based on the Cartesian product as follows:

$$\ell = \prod_{x \in P} |Outcomes(x)| \tag{2}$$

Some of these levels will be unusable, for example, fully correct syntax and semantics can't produce fully incorrect results. For any platform, the following procedure shows how to instantiate the generic model:

1. Choose the properties P that you want applied in the grading system and possible outcomes T.
2. For each $x \in P$, choose $Outcomes(x)$.
3. Determine inappropriate levels and remove them.

2.6 Flexibility in Adding or Removing Properties

Our model is based on having a correctness matrix that comprises of all the combinations of the given properties and their outcomes. This matrix is shown here as a table for example, a matrix of results, semantics and syntax each having outcomes: correct and incorrect is shown in Table 1. The final grade is calculated from this matrix as we will see in Subsect. 3.3. To carry out binary grading, we would generate a correctness matrix using only results with the outcomes correct and incorrect. This would translate in the correctness model having two levels: level 1 for an incorrect query and level 2 for a correct query. Using the results, we can also generate a correctness model of 3 levels. The results would have

the following levels and outcomes: level 1 for fully incorrect results, level 2 for results containing the expected results within and level 3 for fully correct results. Adding a property or an outcome increases the size of the correctness matrix which also increases the number of correctness levels as well as the sensitivity of the model. Similarly, removing a property or an outcome decreases the number of correctness levels, which then reduces the sensitivity of the model. The number of properties or outcomes to use depends on the sensitivity required by the instructor. Having the correctness matrix as the base of grading makes it easy to incorporate or remove additional properties during grading.

3 Method

This section describes how we used the correctness model for grading. In order to accomplish this, we implemented an automated grading program for SQL queries. The program was designed using C++ and utilized PostgreSQL database management system under Ubuntu as well as Debian. The following are the steps we took to accomplish this.

3.1 Choosing Properties and Outcomes

To assess the quality of SQL queries written by students, there are a variety of properties we could have chosen. We decided to focus on three key properties, syntax, semantics, and results. The pool from which we chose these properties is: syntax, semantics, results, complexity, efficiency, maintainability, reliability, scalability, usability, portability, security, interoperability, testability, completeness, reusability, robustness, modularity, adaptability and understandability. We checked syntax to ensure students have a basic understanding of how SQL is written. Evaluation of semantics tested whether students could take a given task and transform it into written code. Assessing results confirmed students could write queries that delivered the required results. By focusing on these properties, we could provide an accurate evaluation of a students' understanding and application of SQL. We did not consider properties such as complexity and efficiency as database management systems feature query optimizations [4].

We generated our outcomes by simply subdividing the group containing incorrect queries. Some of the outcomes we could have used are shown in Fig. 1, in which sensitivity is increased from left to right. In the figure, incorrect means completely incorrect. The level of sensitivity for a given property depends on the goals of the instructor. We decided to go as far as minor incorrect for each property. The meaning of each group of outcome will be explained in the coming subsections.

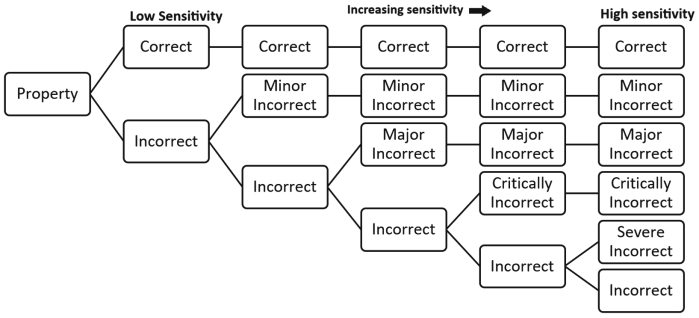

Fig. 1. Some outcomes that can be used during the assessment of various properties.

3.2 SQL Queries Outsourcing

We experimented with correctness models for an automatic grading system of SQL exercises in the first year's BSc course, Information Modelling and Databases at Radboud University. This course, at the time of our study, had 375 students enrolled. Previously, several SQL quizzes had already been provided to the students and graded in a binary approach. For our study, we selected 6 questions from these quizzes. These questions contained 2,113 students' answers that served as our starting point. Each question, containing more than 300 SQL queries was then exported as a CSV file as well as the instructor's model correct answers in a separate CSV file. The implemented program that ingested these queries followed the procedure outlined below.

3.3 Program Input and Preprocessing

The program received the name of a CSV file containing a list of students' SQL queries for a single question and the name of a CSV file containing the instructor's model correct answers. The instructor is free to add as many model answers as possible. Nevertheless, the given model answers may not be enough to encompass all the possible correct query formulations. The relevant schemas had already been created in the database management system.

Following this, the student queries were saved in a data structure that would allow fast access to a query at any time during processing. Each query in the data structure had a unique ID and a query string. We removed unnecessary spaces from each query during the process of saving the query in the data structure. We also created a copy of the data structure containing the queries. This was to keep track of the edit changes made to a query. Afterwards we saved the expected correct queries in a different data structure.

Lastly, we initialized 4 grading parameters: the number of syntax, semantic and results outcomes, and a variable dictating the order of importance of these three properties. The number of outcomes was limited to 2 or 3 for each property. Value 2 meant the assessment would check if the property was either correct or

not. Value 3 meant the property was to be categorized as correct, minor incorrect or fully incorrect. The 'order of importance' variable established which property had more weight during grading. For this study, the variable was set to prioritize results the most, followed by semantics and finally syntax. We then instantiated a correctness model using these parameters. A correctness model with the three parameters set to 2 is shown in Tables 1, 2 and 3. These tables show 8 Magnitude correctness levels of varying syntax, semantics, and result, and the effects of the weight variable on a specific outcomes group. A table with parameters set to 3 would contain $3^3 = 27$ outcomes.

The Property Weight Variable. The weighing variable introduced here changes the correctness level in which a specific outcomes group is placed. The variable alters the correctness model as shown in Tables 1, 2 and 3. For example, when syntax is prioritized, having correct syntax jumps the level of the query all the way to correctness level 5 as shown in Table 3. If query 2 shown below has the wrong semantics and results, the fact that it has the correct syntax puts the query in level 5. This is in contrast to query 1 which has a simple error in 'SELECT' therefore it would be placed in level 1.

Table 1. Results has more weight followed by semantics and then syntax

Level	Results	Semantics	Syntax
1	incorrect	incorrect	incorrec
2	incorrect	incorrect	correct
3	incorrect	correct	incorrec
4	incorrect	correct	correct
5	correct	incorrect	incorrec
6	correct	incorrect	correct
7	correct	correct	incorrec
8	correct	correct	correct

Table 2. Semantics has more weight followed by syntax and then results

Level	Results	Semantics	Syntax
1	incorrect	incorrect	incorrec
2	correct	incorrect	incorrec
3	incorrect	incorrect	correct
4	correct	incorrect	correct
5	incorrect	correct	incorrec
6	correct	correct	incorrec
7	incorrect	correct	correct
8	correct	correct	correct

Table 3. Syntax has more weight followed by semantics and then results

Level	Results	Semantics	Syntax
1	incorrect	incorrect	incorrect
2	correct	incorrect	incorrect
3	incorrect	correct	incorrect
4	correct	correct	incorrect
5	incorrect	incorrect	correct
6	correct	incorrect	correct
7	incorrect	correct	correct
8	correct	correct	correct

```
Query 1:
SELEXT * FROM my_table WHERE mycol = 1;
Query 2:
SELECT * FROM my_table WHERE mycol = 1;
```

Query 2 would be placed in level 3 when semantics are weighed more as shown in Table 2. Weighing more on syntax can be used in an introductory class whereby, mastering SQL syntax would be the primary goal. Similarly, weighing more on semantics can be used in an intermediary class whereby, the focus would be mastering how to carry out the given SQL tasks based on the specifications. Finally, weighing more on results can be used in an advanced class whereby, the end results would be more important.

The property weight variable affects the grade awarded to a query since the grade is dependent on the correctness level. The grade is calculated using Eq. 3.

$$g = \frac{l - min(l)}{max(l) - min(l)} \tag{3}$$

where g grade in the range 0 to 1, l is the correctness level, $min(l)$ is the minimum correctness level and $max(l)$ is the maximum correctness level of a specific model in use.

3.4 Result Analysis

The aim of this stage was to create an initial pool of queries with correct results and those with not. A query with correct results would have correct syntax as well. The *results* property was checked by selecting data from the example database using the student's answer, and comparing that to the data selected by the teacher's model answer. Initially, we created a data structure to house correct queries. Model correct queries were added to this data structure to be used later to carry out edit distance comparisons. We then executed one of the instructor's model correct answers and stored the output in another data structure. Afterwards, we executed each student query and compared the output with the expected results. The correct student queries were added to the correct queries data structure. By doing so, all correct student queries would be used to assess the correctness levels of the rest incorrect student queries. All the queries were given the appropriate tag showing the resulting outcome. A student query with correct results is the one that produces the same results as the model query across various datasets. In this implementation, we used one dataset, but we plan to incorporate additional datasets in our next implementation.

The outcome for each query was dependent on the variable that was set in Subsect. 3.3. For two outcomes, a query was tagged as either correct or incorrect. For three outcomes, the query was tagged as correct, minor incorrect or fully incorrect. A query with minor incorrect results meant that results of the model query were a subset of the students query results. For instance, a missing WHERE clause would sometimes produce unfiltered data.

3.5 Syntax Analysis

In this stage, we checked if a query was written with the correct syntax. To accomplish this, we checked if the query could be correctly parsed by the PostgreSQL parser. For this purpose, we utilized the library, pg_query [11]. This library is able to create abstract syntax trees of queries. All the queries were parsed taking note of the queries that were parseable and those that were not. They were then tagged with the appropriate outcome.

The outcome for each query was also dependent on the variable that was set in Subsect. 3.3. For two outcomes, a query was tagged as either having correct or incorrect syntax. Three outcomes required edit distance comparison

with the pool of correct queries that was populated in the previous step. The assumption in this case was that all correct queries had correct syntax. For any incorrect query, edit distances were calculated between the query and the pool of correct queries containing instructor's model correct answers as well as correct student queries. The lowest edit distance was noted. If the edit distance was less than three characters, the syntax was deemed minor incorrect. We felt that setting the threshold to two characters covered simple mistakes such as a wrong character in a keyword or a missing comma. We plan to experiment on different thresholds in our future work. Next, the query was automatically adjusted based on the closest correct query. In all other cases, the syntax was considered fully incorrect. Levenshtein's edit distance [17] was used for comparisons.

3.6 Semantic Analysis

The final stage of our study involved semantic analysis. In this context, semantics error meant any error that occurred beyond the parsing stage. Examples of these errors are wrong table names, columns and join clauses. Semantic analysis was accomplished by comparing parse tree edit distances between correct queries and those that were tagged as incorrect in earlier stages. We accomplished this comparison using Shasha Zhang's algorithm [19].

The outcomes were allocated similarly to syntax analysis. This was accomplished by comparing the tree edit distance between a query and correct queries. However, in this stage, minor incorrect semantics stood for edit distances less than two. In all other cases, including those queries that were not parseable, semantics was considered as completely incorrect.

3.7 Grading

At this stage, the queries have already been tagged with various outcomes for syntax, semantics and results. We carried out grading based on various configurations for the property outcomes, including for example binary grading as well as more sensitive grading. Finally, we tagged a query with the appropriate correctness level based on the correctness model we instantiated in Sect. 3.3. We saved analysis information to an external CSV file.

4 Results

In our exploration of how sensitivity of a grading system affects grading, we graded 2,113 SQL queries, given by 375 students for 6 different questions. This was accomplished by varying the number of properties checked and their outcomes and our findings are summarized in Fig. 2. From left, the first bar plot uses only syntax for grading. The second bar plot uses syntax, semantics and results each evaluated as either correct or incorrect. The last bar plot uses the three properties with each evaluated as correct, minor incorrect or incorrect.

The scatter plot on the left shows the groups created when grading using an 8-magnitude model. The right scatter plot shows the groups formed after grading using a 27-magnitude model. The values in parentheses show the outcomes for syntax, semantics and results respectively. For 8-magnitude: 0 means incorrect and 1 correct. For 27-magnitude: 0 means incorrect, 1 minor incorrect and 2 correct.

Initially, grading the queries using a 2-magnitude correctness model (1-property, 2-outcome correctness model, where the property is "Results") yielded two groups of queries: correct and incorrect queries. There were 1071 queries that were deemed correct and were therefore labelled as level 2. In contrast, 1038 queries were categorized as incorrect, i.e., level 1. We observed that queries with a different and varying number of errors were categorized as one group.

Subsequently, we observed that grading the queries using an 8-magnitude correctness model produced more groups than in a 2-magnitude model. There were 4 distinct groups of graded queries, including the following categories:

- 1071 answers were fully correct.
- 851 answers managed only correct syntax.
- 187 answers were incorrect in all the properties.
- 4 answers made a negligible mistake in syntax. We may have missed those student queries with small syntax mistakes whose syntax had no close comparisons. This is because our syntax analysis employed comparisons between students queries and model queries and also other correct student queries.

Specifically, 3 of these groups were formed as a result of the current model differentiating the single group of incorrect queries from the 2-magnitude model. Meanwhile, the group containing the correct queries retained the same number of queries. In this model, the group occupied the highest correctness level, 8.

Lastly, grading the queries using a 27-magnitude correctness model yielded the largest number of distinct groups of graded queries. There were 9 distinct groups of graded queries including the following categories:

- 1071 answers were fully correct.
- 556 answers managed only correct syntax. For example,

```
Student query: SELECT name From Artist, Produces;
Model Query: SELECT DISTINCT name FROM Artist, Produces
    WHERE Artist.artist_id = Produces.artist_id;
```

Even though the query could be parsed, it did not carry out the intended task.

- 290 answers managed only correct syntax with minor incorrect results. For example,

```
Student query: SELECT A.theme_id, A.name FROM Theme A,
    Teacher B WHERE B.name = 'Djoerd Hiemstra'
Model Query:   SELECT T.theme_id, T.name FROM Theme T,
    Teacher N  WHERE N.name='Djoerd Hiemstra' AND T.
    teacher_id=N.teacher_id;
```

The query did not filter the results correctly.

- 161 answers were incorrect in all the properties.
- 23 answers had minor incorrect syntax. The rest of the properties were incorrect. For example,

```
Student query: SELECT Artist.name FROM Artist, Produces;
     WHERE Artist.artist_id = Produces.produces_id;
Model Query: SELECT DISTINCT name FROM Artist, Produces
    WHERE Artist.artist_id = Produces.artist_id;
```

The student query had the wrong condition in the filter clause and also did not include keyword DISTINCT.

- 4 answers had partially correct semantics and results. For example,

```
Student query: SELECT album_title FROM Album WHERE type=
    'complilation';
Model Query:   SELECT album_title FROM Album WHERE type=
    'compilation';
```

The current implementation of our model deducted points from results when semantics was minor incorrect. In the example, the student query had a misspelled filter condition. The program deducted points for the small mistake in semantics and also for the results. In our next version of the implementation, we are considering deducting only semantics points for such cases.

- 4 answers had only a negligible mistake in syntax. For example,

```
Student query: SELECT album_title; FROM Album; WHERE
    Album.type='compilation';
Model Query: SELECT album_title FROM Album WHERE type='
    compilation';
```

The query had extra characters, ';' which made the syntax minor incorrect.

- 3 answers had partially correct syntax, semantics and results.
- 1 answer made a minor mistake in semantics resulting in incorrect results.

As previously observed, the group containing the correct queries still had the same number of queries. Notably, in this model, the group occupied the highest correctness level, 27.

The groups seen were not spread out well among the possible levels. This comes from three reasons. First, some levels are unusable. For example, both syntax and semantics cannot be fully incorrect while the results, fully correct. These levels were left unoccupied. Second, the mistakes found in the queries were also not evenly distributed. Changing the threshold for minor syntax and semantics errors would affect the groupings seen. Third, syntax and semantics analysis were sometimes entangled in our implementation. When both semantics and syntax error were present in the student query, automatic editing in the

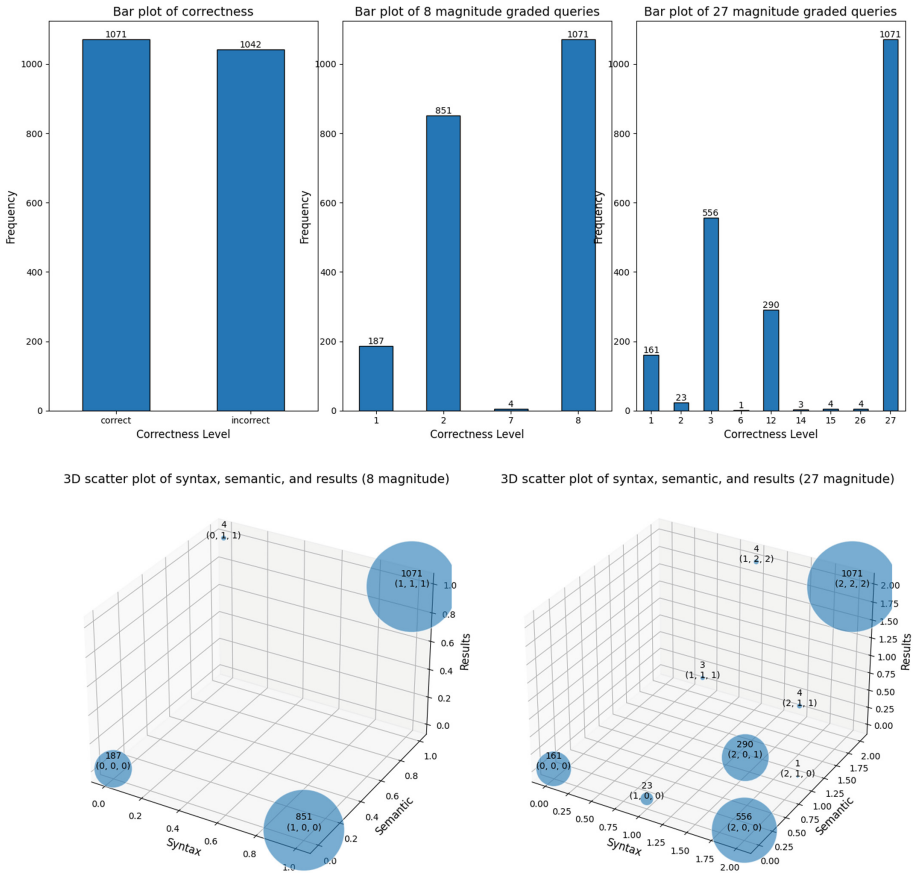

Fig. 2. Distinct group categorization of SQL queries across varying correctness model magnitudes.

syntax analysis stage sometimes also fixed the semantic errors. We are working on a complete separation between syntax and semantics analysis for our next implementation to fix the third reason and a better distribution of grades along the possible correctness levels.

5 Conclusion

5.1 Evaluation Outcomes

By increasing the number of properties and the number of outcomes in the grading process, the number of students that received a (fully) "incorrect" grade decreased from 1042 students to 161 students. Concurrently, differentiation of incorrect answers increased. This method enabled us to better identify the well

written parts of the query and those that were lacking. Consequently, we could effectively award grades and group queries based on correct or incorrect features.

This study demonstrates how correctness levels can be used to grade students SQL queries. The correctness model allows for flexibility in adjusting the number of properties to be used during grading and their outcomes. First, it allows coarse grading by using only the property results with outcomes, correct and incorrect. This might be used when we want to emphasize the importance of correct results for critical requirements in advanced classes. In this case, high marks are awarded when the query completely satisfies the given requirements. For example, exercises for practising data query for hospital scenarios would call for strictness in grading to emphasize how important it is for such a system to return the correct patient records. Second, the model also allows fine grained analysis by incorporating more properties and outcomes. Throughout the learning period, acknowledging the parts whereby the students do well and pointing out the missing parts helps the students learn better, while as well as protecting their motivation. Fine grained analysis enables this. Third, it allows for giving more weight to a specific property during grading, for example, syntax. This helps to focus on a specific goal at a certain period of the teaching process. Like, writing the correct SQL syntax in this case. Lastly, the model allows for cases between these two given extremes. All of these features enable better provision of grades based on the objectives of the instructor.

5.2 Grading Results

The results of this study, also demonstrate that we can enable an automated grading system for SQL to categorize students answers into distinct groups. Categorization is achieved through independent evaluation of properties such as syntax, semantics and results, along with their corresponding outcomes including correct, minor incorrect and fully incorrect. The number of groups produced can be varied by adjusting the sensitivity of the grading system. This can be achieved in two ways. Firstly, we can vary the number of properties being checked. The inclusion of syntax, semantics, and results in the evaluation process will yield more groups compared to solely checking results. Secondly, we can vary the number of outcomes for each property. By checking whether the syntax is correct, minor incorrect or fully incorrect, we can produce more groups than if we were to only consider correct and fully incorrect outcomes.

5.3 Implications

An implication of this study is that we can protect and further improve students' motivation to learn simply by awarding a grade that considers both their effort and level of understanding, even if the final answer might not be correct. Furthermore, we are able to vary grading checks to suit the goals of the instructor at a specific point in the teaching process. Moreover, the ability to adjust the sensitivity of the grading system allows the instructor to conduct a fine-grained analysis in order to recognize what specific students need in terms of learning

materials. Consequently, instructors can provide better learning materials that match the educational needs of a specific group of students. We argue that incorporating more correctness levels in the grading system achieves a more detailed and accurate evaluation of what a student understands and what is lacking. This stands in contrast to simply grading a student's answer as either correct or incorrect, which may fail to notice and appreciate what the student has gotten right.

5.4 Building on Prior SQL Studies

Our study corroborates the research conducted in SQLify [6]. The authors managed 8 correctness levels to group and grade answers given by students of which 3 levels are manually graded. Our model is able to achieve more automatically graded levels. In other studies, fixed procedures were employed during evaluation limiting the inclusion of additional properties [3,5,7,8]. In contrast, our method offers a dynamic and procedural approach of adding or removing correctness levels depending on the goals of the instructor. The flexibility in adjusting the number of correctness levels enables personalized grouping of students' answers, allowing instructors to decide on groupings that align with their goals.

5.5 Limitations

A limitation of this study is that we went as far as considering minor mistakes made by students for different properties. Adding more outcomes for example, major incorrect and critically incorrect would produce a better fine-grained assessment. Second, provision of sample data can help students formulate queries that might produce the correct results but have incorrect semantics. However, we were able to demonstrate how the sensitivity of an automated grading system can be varied resulting in different grouping of students' answers. We were able to effectively categorize students into groups that show their corresponding strengths and weaknesses by going as far as recognizing their minor mistakes. In our future work, we plan to expand the analysis by increasing the number of outcomes checked for each property, aiming for an even more fine-grained analysis of students' SQL queries.

5.6 Benefits

In conclusion, this study has shown that adjusting the sensitivity of automated SQL grading in computer science courses can aid in addressing the educational needs for both students and instructors. For the students, identification and rewarding for the things they have gotten right in partial grading becomes possible. Additionally, for the instructors, automatically providing better grades based on their objectives and grouping students based on their strengths and weaknesses as shown by the given answers becomes possible. As a result, the educational needs for each categorized group can now be addressed by offering specialized learning materials.

References

1. Ala-Mutka, K., Uimonen, T., Järvinen, H.-M.: Supporting students in C++ programming courses with automatic program style assessment. JITE **3**, 245–262 (2004)
2. Bhangdiya, A., et al.: The XDa-TA system for automated grading of SQL query assignments. In: 2015 IEEE 31st International Conference on Data Engineering, pp. 1468–1471 (2015)
3. Chandra, B., Banerjee, A., Hazra, U., Joseph, M., Sudarshan, S.: Automated grading of SQL queries. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 1630–1633 (2019)
4. Chaudhuri, S.: An overview of query optimization in relational systems. In: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '98, pp. 34–43, New York, NY, USA. Association for Computing Machinery (1998)
5. Dambić, G., Fabijanić, M., Ćošković, A.L.: Automatic, configurable and partial assessment of student SQL queries with joins and groupings. In: 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO), pp. 837–842 (2021)
6. Dekeyser, S., de Raadt, M., Lee, T.Y.: Computer assisted assessment of SQL query skills. In: Proceedings of the Eighteenth Conference on Australasian Database - Volume 63, ADC '07, pp. 53–62, AUS, 2007. Australian Computer Society, Inc. (2007)
7. Fabijanić, M., Dambić, G., Sasunić, J.: Automatic, configurable, and partial assessment of student SQL queries with subqueries. In: 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), pp. 542–547 (2022)
8. Kjerstad, J.: Automatic evaluation and grading of SQL queries using relational algebra trees. Master's thesis, Norwegian University of Science and Technology (2020)
9. Kleiner, C., Tebbe, C., Heine, F.: Automated grading and tutoring of SQL statements to improve student learning. In: Proceedings of the 13th Koli Calling International Conference on Computing Education Research, Koli Calling '13, pp. 161–168, New York, NY, USA, 2013. Association for Computing Machinery (2013)
10. Nalintippayawong, S., Atchariyachanvanich, K., Julavanich, T.: Dblearn: adaptive e-learning for practical database course - an integrated architecture approach. In:2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp. 109–114 (2017)
11. T. Pganalyze Developer Team. libpg_query, 2023. Version 15-4.2.1
12. Prior, J.C., Lister, R.: The backwash effect on SQL skills grading. In: Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE '04, pp. 32–36, New York, NY, USA, 2004. Association for Computing Machinery (2004)
13. Tallent, N.R., Mellor-Crummey, J.M.: Effective performance measurement and analysis of multithreaded applications. In: PPoPP '09, pp. 229–240, New York, NY, USA, 2009. Association for Computing Machinery (2009)
14. Wanjiru, B., Bommel, P.V., Hiemstra, D.: Towards a generic model for classifying software into correctness levels and its application to SQL. In: 2023 IEEE/ACM 5th International Workshop on Software Engineering Education for the Next Generation (SEENG), pp. 37–40 (2023)

15. Wilkie, F.G., Hylands, V.: Measuring complexity in C++ application software. Softw. Pract. Exp. **28** (1998)
16. Yoshida, H., Tokumoto, S., Prasad, M.R., Ghosh, I., Uehara, T.: FSX: fine-grained incremental unit test generation for C/C++ programs. In: Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, pp. 106–117, New York, NY, USA, 2016. Association for Computing Machinery (2016)
17. Yujian, L., Bo, L.: A normalized levenshtein distance metric. IEEE Trans. Pattern Anal. Mach. Intell. **29**(6), 1091–1095 (2007)
18. Zehra, F., Javed, M., Khan, D., Pasha, M.: Comparative analysis of C++ and python in terms of memory and time, December 2020
19. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM J. Comput. **18**, 1245–1262 (1989)