Partial SQL Query Assessment

Mario Fabijanić*, Igor Mekterović**
*Algebra University College/Software Engineering, Zagreb, Croatia mario.fabijanic@algebra.hr
**Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia igor.mekterovic@fer.hr

***Abstract* – Automated grading systems in education have been around for sixty years. They have found applications in areas such as online learning systems where virtually an unlimited number of users can test their knowledge that would not be possible to evaluate manually. Implementations within Massive Open Online Courses are a good practice in which users can do the self-testing, and get instant feedback, making the learning process more efficient. Within universities, automated grading systems allow teachers to evaluate solutions and provide feedback for thousands of submissions in a short time. This paper presents an overview of methods used in automatic SQL query evaluation systems, from early implementations when the goal was only to evaluate solutions binary, to today when they enable functionalities like partial and configurable evaluation, rich and customized feedback, learning analytics, learning pattern detection, code quality check, plagiarism detection. These methods are not exclusive, and combining different approaches makes an automated grading system more comprehensive and applicable. Automatic assessment system of SQL queries developed at Algebra University College will be presented as an example of a system which uses dynamic and static analysis, awards partial points, and gives feedback to students based on the wrong parts of their solutions.**

Keywords – SQL; assessment; fully-automated; partial marks; the configurable assessment process

I. Introduction

The rising demand to educate IT professionals increases the burden on teachers who, during the educational process, often must quickly and fairly evaluate students' solutions in the form of programming language code. The problem was recognized more than 60 years ago [1] and is present nowadays more than ever in IT colleges and universities. Online assessment systems in education have been used since 1961 when one of the first systems was used at Stanford University for the evaluation of submissions written in ALGOL [2].

In database-related university courses, the main problem is the amount of time required to evaluate student solutions. The assessment process imposes a cognitive load that lasts for days and can result in delays and inconsistencies [3]. For instance, teachers included in research [3] spent 22 hours grading 1533 student SQL solutions, and the assessment process was not finished, since students successfully challenged given grades (66% of given grades were changed). Furthermore, only 33% of teachers finished the assessment process before the deadline defined by their university. This problem is often mitigated by involving additional teachers in the assessment process, but that may lead to diminished consistency and favoritism [4]. This additional inconsistency when more teachers are included in the grading process is more emphasized especially when partial grades are assigned because teachers can have different grading criteria [5].

Manual feedback from the teacher helps student performance [6][7]. Thus, automatic feedback becomes an essential part of the education process, because often during the assessment process, an insufficient number of students get feedback that would help them fix the score next time [8]. This precise feedback is often not provided or is not available to most students. Automatic instant feedback increases students' participation and helps them resolve assignments in introductory courses [9], with improved performance and final grades [10]. Automatic instant feedback can also be implemented in a system used to learn how to construct SQL queries, providing tips during writing queries [11]. It also helps teachers to increase teaching efficiency, with a reduced workload [12]. As will be shown in the remainder of the paper, automatic grading and feedback generation are closely related.

II. Grading SQL Statements

This chapter focuses on techniques and methods currently used or proposed by researchers in the field for evaluating SQL (Structured Query Language) code used in relational database systems. Semi-automated systems provide only part of the whole grade, and teachers do the final grading, while in case of fully automated systems, teachers do not need to evaluate individuals' solution, but they need to setup assessment system. Many semi-automated or fully-automated assessment systems were introduced during the years [15][16][17][18][19][20][21][22]. Those systems are based on dynamic analysis, which was the first method used in evaluation process with automated assessment systems.

A. *Dynamic analysis*

Dynamic analysis is performed by executing student's solution and by executing the correct solution assigned by the teacher. Both solutions use the same input datasets to be able to find the differences in results and give a grade that depends on the result sets' similarity. Dynamic assessment then concludes if the student's solution is fully correct or not. Typically, dynamic assessment implementation produces a binary decision, although, partial gradings are possible as with ActiveSQL but are of questionable application. The first generation of assessment systems implemented dynamic analysis with binary grading, without the possibility of giving partial points [15][24].

Dynamic assessment is very sensitive to differences in the result sets. For instance, a small, irrelevant error like an extra column in a result set causes "wrong solution" outcome. Depending on the database state and the question, sometimes wrong solutions can return the same result set as the correct solution, so we must have a test database and we must pay attention to its content when creating questions.

Authors in [25] used information from the physical query plan of the RDBMS to generate data that will cover each node from the physical query plan represented as an ordered list or relational algebra operations. To have covering data means to have the data adequate to test different situations in the queries. All the predicates from the query should be evaluated as TRUE to get solution data as a result set for the query testing. The goal is to get data that satisfies a coverage criterion and to populate tables with such test data. To achieve this, random search may be used to explore a set of random data, biased random search to explore a set of random data, but with seeded values from the query constants or from the seeding strategy specific for join operations, and the genetic algorithm may use a fitness function for the data from physical query plan that measure how close candidate solution is from covering query target. The idea of generating query-tailored data will improve automated assessment systems making them even more reliable during grading student solutions. Regarding feedback, dynamic analysis cannot provide the reason why the solution is wrong. To overcome that imperfection, some systems are giving feedback based on the comparison of other parameter values like row count [14].

In conclusion, dynamic analysis is a very practical, popular, and widely used technique that is relatively simple to implement. On the other side, it does not provide partial grading facilities and can only provide rudimentary feedback. In a try to overcome that, some of the proposed models provide partial scoring feature comparing students' solution result sets with the expected result set from the correct solution. Partial grade is then calculated as a proportion of correct cell values of the total cell values of the correct result set. However, this is a rather dubious approach with limited applicability [21s]. It should be noted that dynamic assessment attempts to (dis)prove equivalence of two queries by comparing their effects (outputs). Why not compare the queries themselves which is our problem in the first place? The following section describes such approach.

B. *Queries equivalence*

The problem of recognizing SQL query equivalences is not a simple task because of its' undecidability [26]: two SQL queries can be syntactically different, but semantically/logically the same and both will produce the same result set for the same inputs. A group of authors proposed an automated prover which can determine the equivalence of SQL queries, named Cosette [27]. Cosette consists of two components: first component translates SQL queries into logic formulas, and then searches for the examples of input data which would show that the two queries are not the same, and the second component translates queries into K-relations, which are then validated for the equivalence. K-relations are the representation of relations in the form of mathematical functions which receive a tuple and return input tuples' multiplicity, which can be checked to proof the equivalency [27].

Cosette has several drawbacks due to the limitations in HoTT (Homotopy Type Theory) library [35] for Coq proof tool, on which it is based on [26]. For instance, Cosette does not support foreign keys because they are difficult to model with HoTT library in Coq [26]. To overcome this limitation, authors of Cosette proposed new algebraic structure - unbounded semiring (U-semiring). They used it as new formalism for SQL queries. To prove that two queries are semantically equal, SQL queries are converted into U-semiring expressions (U-expressions). In the next step, U-expressions are compared using UDP (U-expression Decision Procedure) algorithm [26]. U-semiring may model some of the SQL features like integrity constraints, which was one of the main drawbacks in Cosette. UDP supports more different SQL queries then Cosette, with more powerful automated proof search [26]. Although these methods brought new algorithms for checking the equality of two SQL queries, and results were promising, to the best of our knowledge, their development for wider range of SQL features didn't happen, probably due to their overall complexity.

C. *Static analysis*

Rather than proving that two SQL queries are equivalent, which has proven to be too difficult [26], many researchers turned to static analysis of SQL statements. It resulted in the second-generation automated assessment systems, which use static analysis method [13]. In situations when the result sets from student solution and correct solution are equal, without static analysis system cannot conclude if students' solution fully satisfies the expected solution. For instance, in case when the assignment (and solution) explicitly calls for using joins, and student provides a logically equivalent solution which uses subqueries.

Static analysis is the analysis of the code without executing it. In our context, it can be used to analyze structure of the SQL code, or its' abstract syntax representation which is known as the abstract syntax tree (AST). By comparing parts of AST, the similarity of SQL queries may be determined [28]. AST represents the code written in a programming language and adjusted to contain only nodes that matter for the comparison, without elements like comments, semicolons, parenthesis, aliases, or any other elements without an impact on the functionality of the solution.

Before creating AST, SQL codes that should be compared are converted into canonical forms. Canonical form of SQL query is again a SQL query, but without parts that are irrelevant for the comparison with another query [29]. Canonical forms of SQL queries minimize the syntactic differences between two queries that need to be compared. After the process of canonicalization, abstract trees are created from canonical forms.

i) *Canonicalization process*

The process of canonicalization can be divided into syntactical and semantic canonicalization. Examples of syntactical canonicalization challenges:

1. When the solution uses the BETWEEN predicate, it is replaced with the equivalent relational operators "greater or equal to" and "smaller or equal to", e.g.,

```
A BETWEEN 5 AND 10
```

is replaced with

`A>=5 AND A<=10` [29].

2. When the solution uses NOT, it is converted into a statement without NOT, e.g., NOT (A>B) is replaced with A<=B. In addition, predicates that contain greater than are replaced with smaller then, e.g., A>B is replaced with B<A [29].

3. When the solution uses a subquery with the ORDER BY clause and without the TOP clause, the ORDER BY clause is removed because it does not change the result set.

4. When the solution uses INNER JOIN, the query optimizer chooses optimal order during query execution [30], and it can be different in the correct solution than the order in the student solution. For instance, when comparing two statements:

```
FROM City as c INNER JOIN Student as s
 ON c.CityID=s.CityID
```

and

```
FROM Student as s INNER JOIN City as c
 ON c.CityID=s.CityID
```

Regarding functionality, those two statements are equal, but their abstract trees are different, and that is why those kinds of nodes are sorted before calculating the similarity of statements.

Examples of semantic canonicalization challenges:

1. When the solution uses the DISTINCT clause to remove duplicates, and they do not exist, the DISTINCT clause can be removed from the solution. The case of the primary key attribute or any other unique attribute in the SELECT list and as a join predicate, meaning the DISTINCT clause can be removed [29].

2. When a solution contains redundant joins, they can be removed, e.g., if the solution is:

```
SELECT * FROM student
LEFT OUTER JOIN department
ON student.dept_id=department.id
WHERE student.dept_name = 'Biology'
```

The condition in the WHERE clause fails if student_dept_name is null, and the solution can be rewritten to use INNER JOIN instead.

Left or right outer join can be replaced with an inner join if (i) the attributes used in the join include all foreign key references from one side of a join to the other side of a join and (ii) foreign key values used in the join are not null [29].

3. When a solution contains the predicates which are replaceable, lexicographically least attribute will be used in all clauses, e.g., if the query is:

```
SELECT student.dept_name FROM student
INNER JOIN department
ON student.dept_name=department.dept_name
WHERE student.dept_name LIKE 'English%'
```

And because of the student.dept_name=department.dept_name, the department.dept_name attribute will be used in the SELECT clause too. In addition, if constant is a part of the equality condition, it is considered as lexicographically least attribute [29].

4. When a solution contains an ORDER BY clause with more than one attribute, and if the second attribute is functionally dependent on the first attribute, the second attribute is ordered the same way as the first attribute, and therefore, irrelevant, and can be removed from the solution. Similarly, when a solution contains GROUP BY, e.g.,

```
SELECT id, COUNT(*) FROM student
INNER JOIN takes ON student.id=takes.id
GROUP BY id, name
```

and if student.id is the primary key, the attribute name can be removed from the GROUP BY clause [29].

ii) Using ASTs

After the canonicalization process, ASTs are used to find the differences between students' solution and correct solution. One possible approach is to calculate tree edit distance between students' solution tree and correct solution tree [28]. It is calculated as number of modifications to the student solution tree until it becomes equivalent to the correct solution tree. This makes partial grading based on the number of needed modifications possible, but the challenge regarding this approach is the need to test the equivalence after each modification of the student solution. There is also a possibility to try all the edits possible at each step, again testing the equivalence at each step, making the number of edits exponential [29]. During tree similarity calculation, ASTs may be decomposed into smaller subtrees which are more suitable for the tree edit distance algorithms.

First tree edit algorithms proposed in the late seventies used a hard-coded strategy and did not consider the shape of the trees which resulted in big differences during runtime [31][32]. More recent algorithms like RTED (Robust Tree Edit Distance) first calculate the decomposition strategy and then adapt to the input tree, which results in better runtime [33]. It is not perfect in terms of memory consumption for large input trees, since the strategy computation can use twice the memory used for distance computation, limiting its usage on large trees. Newer algorithm, AP-TED (All Path Tree Edit Distance) and AP-TED+ reduce the memory consumption by 2/3 compared with the RTED algorithm [33]. This is achieved by releasing the memory early during decomposition.

In addition, the AP-TED+ algorithm considers the fact that during the distance computation phase majority of subtrees contain up to two nodes, and the algorithm uses different functions to compute the distance when one of the trees is small [33]. Another approach of using ASTs is to check a set of grading rules against key-value pairs for each SQL clause within AST. Grading rules make partial grading possible, and they should be created by teachers, together with the correct solution, before assessment process starts.

iii) Without ASTs

The abstract tree can be built when the SQL solution is syntactically correct. If that is not the case, the similarity between two SQL statements should be calculated differently. For that purpose, text comparison of both SQL statements may be done. Before the comparison, both solutions are normalized: tab, newline, and semicolon characters are replaced with whitespace, leading and trailing whitespaces are removed, and all double spaces are replaced with one whitespace. With those adjustments, the similarity of the two texts may be calculated using e.g. Levenshtein distance metric [28]. Levenshtein distance is calculated as the minimum number of single-character edits needed to transform one text into another [34]. Some researchers suggest value 1 as acceptable Levenshtein's distance when a student can still get the maximum grade, and it is considered as an unintentional typo that does not affect student knowledge and therefore, student grade.

D. Combined Approach

Two main approaches are dynamic and static analysis [28], but combining them increases the quality of the automated assessment systems in general. Putting dynamic and static analysis into relation, the result of the dynamic analysis can be one of those:

a) Candidates for correct statements

b) Statements that cannot be executed

c) Partially correct statements.

The dynamic analysis may discover candidates for correct solutions, if they can be executed and provide the expected result set. The problem is that student could get correct result set, but without the need to use expected SQL clauses, and dynamic analysis cannot discover it. Example of this problem is when the UNION statement is used, instead of expected SQL clauses, and tweaks the result set knowing what correct result set should be.

The static analysis is primarily used for partially correct statements, but it can also be used for students' solutions that dynamic analysis confirmed as potentially correct, preventing the fraud like in previous example. In addition, queries that are different in their syntax can be semantically equal, which makes the problem more complex [33].

III. Automatic assessment system of SQL queries developed at Algebra University College

In this chapter, an automatic assessment system of SQL queries developed at Algebra University College will be presented as an example of a system which uses dynamic and static analysis, and therefore may award partial points and give feedback to students, focused on the wrong parts of their solutions. This system uses an input from the teacher: the correct SQL query solution, and the grading rules with the deducting percentage of maximum points in case of errors. During static analysis, the AST of student solutions are compared with AST of the correct SQL query solution. Figure 1 shows an example of the abstract tree application and Figure 2 shows an example of the feedback to student, both taken from an automatic assessment system used during previous exam terms.

Example of the correct SQL query solution:

```
SELECT YEAR(i.InvoiceDate) as InvYr
 , MONTH(i.InvoiceDate) as InvMnth
 , COUNT(i.IDInvoice) as NumberOfInvoices
FROM Invoice as i
 INNER JOIN Customer as c
 ON i.CustomerID=c.IDCustomer
INNER JOIN City as ct
 ON ct.IDCity=c.CityID
WHERE ct.Name='Rijeka'
GROUP BY YEAR(i.InvoiceDate)
 , MONTH(i.InvoiceDate)
HAVING COUNT(i.IDInvoice)<30
   ORDER BY InvYr DESC, InvMnth
```

Example of students' solution:

```
SELECT YEAR(i.InvoiceDate) as InvYr
 , COUNT(i.IDInvoice) as NumberOfInvoices
FROM Invoice as i
INNER JOIN Customer as c
 ON i.CustomerID=c.CityID
INNER JOIN City as ct
 ON ct.IDCity=c.CityID
WHERE ct.Name='Split'
GROUP BY YEAR(i.InvoiceDate)
 , MONTH(i.InvoiceDate)
HAVING COUNT(i.IDInvoice)<30
   ORDER BY InvYr
```

Examples of the grading rules:

a) Regarding the FROM clause:

   i) The weight in the whole query: 100%

   ii) If valid, deduct 0% of possible points

   iii) If not valid, deduct 100% of possible points

b) Regarding the WHERE clause:

   i) The weight in the whole query: 50%

   ii) If not valid, deduct 50% of possible points

   iii) If valid, but wrong columns are included, deduct 25% per each wrong column

   iv) If valid, with correct columns included, but with wrong values included, deduct 25% per wrong value

c) Regarding SELECT clause:

   i) The weight in the whole query: 30%

   ii) If not valid, deduct 10% per each missing column.

Teacher grading rules used during the grading process can be stored in the configuration file or a database. An example of the set of general rules:

```
{ "max_points": 4.0,
  "clause_points_from": 50,
  "clause_points_where": 15,
  "clause_points_groupby": 30,
  "clause_points_having": 15,
  "clause_points_select": 20,
  "clause_points_orderby": 10,
  "parses_and_compiles": 100,
  "wrong_results": 50 }
```

In this example, the JSON configuration file contains the maximum points for the correct solution and weights for each clause in the query. These weights represent percentages that will be deducted from the max_points if erroneous. Also, the rules state the deduction percentage if the solution does not parse and compile, and if it returns a wrong result set. Note that the latter is evaluated via dynamic assessment.

A set of rules per each clause may also be defined:

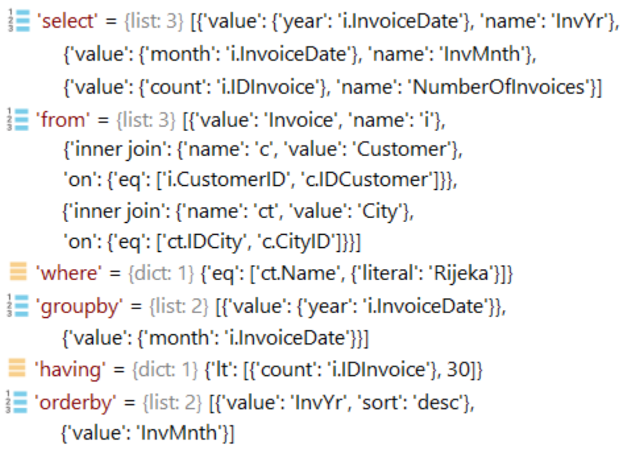
```
"from_each_missing_table"
"from_each_wrong_join"
"from_max_joins_allowed"
"from_use_specific_join"
"where_wrong_rows"
"where_partial_correct_column"
"where_use_specific_operator"
"groupby_not_correct"
"groupby_partial_correct_column"
"having_wrong_number_of_rows"
"having_missing_aggregate_function"
"select_missing_regular_column"
"select_missing_function_column"
"select_missing_aggregate_column"
"orderby_each_missing_column"
"orderby_each_wrong_ascdesc"
"orderby_each_wrong_column_order"
```

An example of the rule for the HAVING clause:

```
"having_missing_aggregate_function":
   {  "status": "on",
      "deduct": 100,
      "hard_stop": 0  }
```

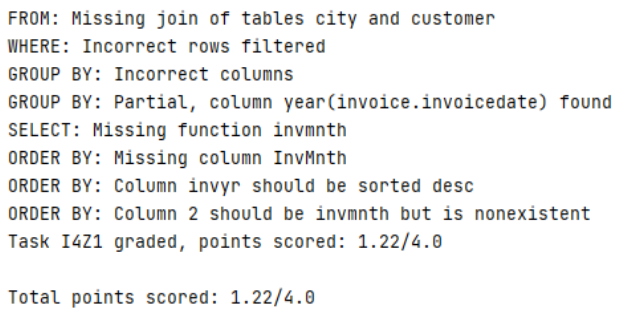
For each of the presented grading rules, there may be a subset of settings that enable or disable the rule, the deduction percentage if the rule is not satisfied, or whether to stop or continue with the grading process, like in above example. Additional settings can also be used, e.g.:

- `specific_join` in the case that student solution must contain required type of inner join, outer join, full join, or cross join in the FROM clause,
- `specific_operator` in the case that specific operator must be used in the WHERE clause,



```
'select' = {list: 3} [{'value': {'year': 'i.InvoiceDate'}, 'name': 'InvYr'},
    {'value': {'month': 'i.InvoiceDate'}, 'name': 'InvMnth'},
    {'value': {'count': 'i.IDInvoice'}, 'name': 'NumberOfInvoices'}]
'from' = {list: 3} [{'value': 'Invoice', 'name': 'i'},
    {'inner join': {'name': 'c', 'value': 'Customer'},
    'on': {'eq': ['i.CustomerID', 'c.IDCustomer']}},
    {'inner join': {'name': 'ct', 'value': 'City'},
    'on': {'eq': ['ct.IDCity', 'c.CityID']}}]
'where' = {dict: 1} {'eq': ['ct.Name', {'literal': 'Rijeka'}]}
'groupby' = {list: 2} [{'value': {'year': 'i.InvoiceDate'}},
    {'value': {'month': 'i.InvoiceDate'}}]
'having' = {dict: 1} {'lt': [{'count': 'i.IDInvoice'}, 30]}
'orderby' = {list: 2} [{'value': 'InvYr', 'sort': 'desc'},
    {'value': 'InvMnth'}]
```

Figure 1. Example of the abstract tree application



```
FROM: Missing join of tables city and customer
WHERE: Incorrect rows filtered
GROUP BY: Incorrect columns
GROUP BY: Partial, column year(invoice.invoicedate) found
SELECT: Missing function invmnth
ORDER BY: Missing column InvMnth
ORDER BY: Column invyr should be sorted desc
ORDER BY: Column 2 should be invmnth but is nonexistent
Task I4Z1 graded, points scored: 1.22/4.0

Total points scored: 1.22/4.0
```

Figure 2. Example of the feedback

- `num` as a number for maximum joins or subqueries needed in student solution,
- `select_subqueries_required` in the case when subquery must be within the SELECT clause,
- `where_subqueries_required` in the case when subquery must be within the WHERE clause.

The precision of the grading rules also has a great influence on the feedback possibilities. Feedback is addressed as one of the most important parts of the whole education process. Rich feedback is a prerequisite that students may become aware of their knowledge gaps and according to those, they can make additional effort in learning [8].

In general, the usability of the feedback is recognized in several areas: (i) info about failure – evaluation logs as the source of that data, (ii) info about failed tests – info about how to proceed, with a hint on how to overcome an issue, (iii) structured report of the evaluation including details, e.g., execution times, resource usage, and tests passed and failed, and (iv) manual report – messages written by the instructor about a submitted solution [13]. With the static analysis implemented, many of the aforementioned ideas for feedback become possible. It is a significant improvement over dynamic analysis, where the feedback was limited to the comments about what is wrong in result set of the query, but without too helpful information about which part of the solution was wrong, and the significance of the error. On top of the partial grading feature, rich feedback is a great improvement in automated assessment systems.

IV. Conclusion

Automated grading systems in the educational process are needed more than ever. Their popularity is growing with the growing number of students enrolled in computer science courses and with the desire to improve the educational process. Partial grading of student SQL queries and rich feedback from the automated grading system could have a major impact on achieving this goal. The main problem to be solved is to prove that two SQL queries (exact solution and student solution) are equal. This is an extremely demanding task, and instead of proving the equivalence of SQL queries, two other approaches that use dynamic and static analysis have prevailed. Dynamic analysis is relatively simple to implement but solves only part of the given problem. On the other hand, static analysis significantly contributes to achieving the set goal - partial evaluation of SQL solutions and at the same time rich feedback for students. Combining both methods greatly increases the capabilities of the system for automatic evaluation of SQL solutions.

References

[1] J. Hollingsworth, “Automatic graders for programming classes,” Commun. ACM, vol. 3, no. 10, pp. 528–529, Oct. 1960, doi: 10.1145/367415.367422.

[2] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, “A survey on online judge systems and their applications,” ACM Comput. Surv., vol. 51, no. 1, 2018, doi: 10.1145/3143560.

[3] J. Tharmaseelan, K. Manathunga, S. Reyal, D. Kasthurirathna, and T. Thurairasa, “Revisit of automated marking techniques for programming assignments,” in IEEE Global Engineering Education Conference, EDUCON, 2021, vol. 2021-April, doi: 10.1109/EDUCON46332.2021.9453889.

[4] D. Fonte, D. Da Cruz, A. L. Gançarski, and P. R. Henriques, “A flexible dynamic system for automatic grading of programming exercises,” OpenAccess Ser. Informatics, vol. 29, pp. 129–144, 2013, doi: 10.4230/OASIcs.SLATE.2013.129.

[5] I. Albluwi, “A Closer Look at the Differences between Graders in Introductory Computer Science Exams,” IEEE Trans. Educ., vol. 61, no. 3, pp. 253–260, 2018, doi: 10.1109/TE.2018.2805706.

[6] D. Nicol and D. MacFarlane-Dick, “Formative assessment and selfregulated learning: A model and seven principles of good feedback practice,” Stud. High. Educ., vol. 31, no. 2, pp. 199–218, 2006, doi: 10.1080/03075070600572090.

[7] J. Hattie and H. Timperley, “The power of feedback,” Rev. Educ. Res., vol. 77, no. 1, pp. 81–112, 2007, doi: 10.3102/003465430298487.

[8] A. P. Cavalcanti et al., “Automatic feedback in online learning environments: A systematic literature review,” Comput. Educ. Artif. Intell., vol. 2, p. 100027, 2021, doi: 10.1016/j.caeai.2021.100027.

[9] S. Krusche and A. Seitz, “ArTEMiS - An automatic assessment management system for interactive learning,” SIGCSE 2018 - Proc. 49th ACM Tech. Symp. Comput. Sci. Educ., vol. 2018-Janua, pp. 284–289, 2018, doi: 10.1145/3159450.3159602.

[10] W. Zhou, Y. Pan, Y. Zhou, and G. Sun, “The framework of a new online judge system for programming education,” ACM Int. Conf. Proceeding Ser., pp. 9–14, 2018, doi: 10.1145/3210713.3210721.

[11] M. H. Ying and Y. Hong, “The development of an online SQL learning system with automatic checking mechanism,” Proc. - 7th Int. Conf. Networked Comput. Adv. Inf. Manag. NCM 2011, pp. 346–351, 2011.

[12] X. Xie and X. Li, “Research on Personalized Exercises and Teaching Feedback Based on Big Data,” ACM Int. Conf. Proceeding Ser., pp. 166–171, 2018, doi: 10.1145/3232116.3232143.

[13] J. C. Paiva, C. I. Tec, and D. C. C. Fcup, “Automated Assessment in Computer Science Education : A State-of-the-Art Review,” pp. 1–39, 2022.

[14] I. Mekterovic, L. Brkic, B. Milasinovic, and M. Baranovic, “Building a comprehensive automated programming assessment system,” IEEE Access, vol. 8, pp. 81154–81172, 2020, doi: 10.1109/ACCESS.2020.2990980.

[15] A. Kleerekoper and A. Schofield, “SQL tester: An online SQL assessment tool and its impact,” Annu. Conf. Innov. Technol. Comput. Sci. Educ. ITiCSE, pp. 87–92, 2018, doi: 10.1145/3197091.3197124.

[16] P. J. Wagner, “The sql file evaluation (sqlfe) tool: A flexible and extendible system for evaluation of sql queries,” Annu. Conf. Innov. Technol. Comput. Sci. Educ. ITiCSE, p. 1334, 2020, doi: 10.1145/3328778.3372599.

[17] A. Trongratsameethong, P. Vichianroj, and C. Mai, “ASQLAG - Automated SQL Assignment Grading System for Multiple DBMSs,” vol. 1, no. 1, pp. 41–59, 2018, doi: 10.14456/jti.2018.4.

[18] C. Kleiner, C. Tebbe, and F. Heine, “Automated grading and tutoring of SQL statements to improve student learning,” ACM Int. Conf. Proceeding Ser., pp. 161–168, 2013, doi: 10.1145/2526968.2526986.

[19] M. De Raadt, S. Dekeyser, and T. Y. Lee, “Do students SQLify? Improving learning outcomes with peer review and enhanced computer assisted assessment of querying skills,” ACM Int. Conf. Proceeding Ser., vol. 276, pp. 101–108, 2006, doi: 10.1145/1315803.1315821.

[20] J. Castelein, M. Aniche, M. Soltani, A. Panichella, and A. Van Deursen, “Search-based test data generation for SQL queries,” Proceedings - International Conference on Software Engineering. pp. 1220–1230, 2018, doi: 10.1145/3180155.3180202.

[21] A. Cumming and G. Russell, “Automatic Checking of SQL: Computerised Grading,” Int. J. Learn. Annu. Rev., vol. 12, no. 3, pp. 127–134, 2006, doi: 10.18848/1447-9494/cgp/v12i03/46714.

[22] J. C. Prior, “Online Assessment of SQL Query Formulation Skills,” Ace ’03, vol. 20, pp. 247–256, 2003, [Online]. Available: http://delivery.acm.org/10.1145/860000/858433/p247-prior.pdf?ip=201.5.254.119&id=858433&acc=PUBLIC&key=4D4702B0C3E38B35.4D4702B0C3E38B35.6DB1341FA924E6B8.4D4702B0C3E38B35&CFID=525851233&CFTOKEN=96683423&__acm__=1436149768_230e7f22b7329beb08e8473b5e50b453.

[23] P. Brusilovsky, M. V. Yudelson, S. Sosnovsky, V. Zadorozhny, D. H. Lee, and X. Zhou, “An open integrated exploratorium for database courses,” Proc. Conf. Integr. Technol. into Comput. Sci. Educ. ITiCSE, vol. 1, no. 412, pp. 22–26, 2008, doi: 10.1145/1384271.1384280.

[24] S. Sadiq, M. Orlowska, W. Sadiq, and J. Lin, “SQLator - An online SQL learning workbench,” SIGCSE Bull. (Association Comput. Mach. Spec. Interes. Gr. Comput. Sci. Educ., vol. 36, no. 3, pp. 223–227, 2004, doi: 10.1145/1026487.1008055.

[25] J. Tuya, M. J. Suárez-Cabal, and C. De La Riva, “Full predicate coverage for testing SQL database queries,” Actas las 16th Jornadas Ing. del Softw. y Bases Datos, JISBD 2011, no. January, pp. 683–684, 2011, doi: 10.1002/stvr.

[26] S. Chu, B. Murphy, J. Roesch, A. Cheung, and D. Suciu, “Axiomatic foundations and algorithms for deciding semantic equivalences of SQL queries,” Proc. VLDB Endow., vol. 11, no. 11, pp. 1482–1495, 2018, doi: 10.14778/3236187.3236200.

[27] S. Chu, C. Wang, K. Weitz, and A. Cheung, “Cosette: An automated prover for SQL,” CIDR 2017 - 8th Bienn. Conf. Innov. Data Syst. Res., 2017.

[28] J. Wang, Y. Zhao, Z. Tang, and Z. Xing, “Combining dynamic and static analysis for automated grading SQL statements,” J. Netw. Intell., vol. 5, no. 4, pp. 179–190, 2020.

[29] B. Chandra, A. Banerjee, U. Hazra, M. Joseph, and S. Sudarshan, “Automated grading of SQL queries,” Proc. - Int. Conf. Data Eng., vol. 2019-April, pp. 1630–1633, 2019, doi: 10.1109/ICDE.2019.00159.

[30] L. Giakoumakis and C. Galindo-legaria, “Testing SQL Server ’ s Query Optimizer : Challenges , Techniques and Experiences,” Bull. IEEE Comput. Soc. Tech. Comm. Data Eng., pp. 1–8, 2008.

[31] E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann, “An optimal decomposition algorithm for tree edit distance,” ACM Trans. Algorithms, vol. 6, no. 1, pp. 146–157, 2009, doi: 10.1145/1644015.1644017.

[32] K. H. Lee, Y. C. Choy, and S. B. Cho, “An efficient algorithm to compute differences between structured documents,” IEEE Trans. Knowl. Data Eng., vol. 16, no. 8, pp. 965–979, 2004, doi: 10.1109/TKDE.2004.19.

[33] M. Pawlik and N. Augsten, “Tree edit distance: Robust and memory-efficient,” Inf. Syst., vol. 56, pp. 157–173, 2016, doi: 10.1016/j.is.2015.08.004.

[34] L. Yujian and L. Bo, “A normalized Levenshtein distance metric,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 29, no. 6, pp. 1091–1095, 2007, doi: 10.1109/TPAMI.2007.1078.

[35] J. Gross, M. Shulman, A. Bauer, P. L. Lumsdaine, A. Mahboubi, and B. Spitters. The HoTT libary in Coq. https://github.com/HoTT/HoTT.