2019 IEEE 35th International Conference on Data Engineering (ICDE)

Automated Grading of SQL Queries

Bikash Chandra*, Ananyo Banerjee†, Udbhas Hazra‡, Mathew Joseph§, S. Sudarshan
IIT Bombay
{*bikash, ananyo16, udbhas16, sudarsha*}*@cse.iitb.ac.in*, mathew_joseph31@yahoo.com

***Abstract*—Grading student SQL queries manually is a tedious and error-prone process. The XData system, developed at IIT Bombay, can be used to test if a student query is correct or not. However, in case a student query is found to be incorrect, there is currently no way to automatically assign partial marks. Manually awarding partial marks is not scalable for classes with a large number of students, especially MOOCs, and is also prone to human errors. In this paper, we discuss techniques to award partial marks to student SQL queries, in case they are incorrect, based on a weighted equivalence edit distance metric. Our goal is to find a minimal sequence of edits on the student query such that it can be transformed to a query that is equivalent to a correct query. Our system can also be used in a learning mode where query edits can be suggested as feedback to students to guide them towards a correct query. Our automated partial marking system has been successfully used in courses at IIT Bombay and IIT Dharwad.**

***Keywords*-SQL query grading, SQL query edits**

I. Introduction

Grading SQL queries is typically done by either by manually checking whether the SQL query submitted by the student matches the correct query or by comparing results of student SQL queries with that of correct SQL queries on one or more fixed datasets. Manual checking of SQL queries is cumbersome and error-prone. Fixed datasets are query agnostic and may fail to catch errors in student SQL queries.

The XData [2], [7] system generates datasets that are tailored to catch common errors for a given SQL query. Datasets generated by XData, based on correct queries, can hence be used to execute the correct queries and student queries, and compare the results. However, for the purpose of grading, just detecting incorrect queries may not be sufficient; it is also necessary to provide partial marks to incorrect queries in such a way that small errors are penalized less than major errors.

A naive approach could be to award partial marks based on the fraction of datasets where the results of instructor query and student query match. However, this approach gives very poor results since student queries may get penalized heavily for a small error (like an error in a selection condition) while student queries that do not even use the correct tables may get some marks since they may pass datasets that produce an empty result on correct queries.

A better way to grade student queries would be to penalize the student based on how many changes are required in the student query to make it equivalent to a correct query provided by the instructor. This allows us to award partial marks in a calibrated manner; a student query that needs more changes can be awarded less marks compared to a student query that needs less changes. However, checking if the edited student query is identical to a correct query is difficult. Many syntactic differences cause no difference in the query result. For example, the selection condition `r.A>5` may also be written as `5<r.A`. Similarly, `ORDER BY id, name` can also be written as `ORDER BY id`, when `id` functionally determines `name`.

We use a variety of query canonicalization techniques to remove many irrelevant syntactic and semantic differences between the student and instructor queries and then compute the edit distance between them. If the edit distance after canonicalization (which we call *canonicalized edit distance*) is 0, the queries are identical. While canonicalized edit distance between student and correct queries could directly be used for partial marking, we show that it has some limitations. Canonicalization and its limitations are discussed in Section II.

In this paper, we propose techniques to award partial marks to a student SQL query based on the query edits required to make it equivalent to a correct query provided by the instructor. The edits could be in the form of insertion, deletion, replacement or movement of parts of the query. The weight of each type of edit can be customized by the instructor. The instructor may provide multiple correct queries. We compute partial marks for the student query with respect to all correct queries and choose the best match i.e. the one that gives the highest marks. We show that the problem of finding the lowest cost edit sequence can be reduced to the problem of finding the shortest path in a graph. We also describe a greedy heuristic technique using canonicalized edit distance that in practice performs well both in terms of runtime and accuracy. We discuss partial marking using query edit based techniques in Section III. Our experiments, described in Section IV, show the effectiveness of our techniques in terms of fairness of the marks awarded.

II. SQL Query Canonicalization

The key idea of this paper is to successively edit a given student query to make it equivalent to a correct query provided by the instructor. However, the same SQL query can be written in multiple correct ways. Our canonicalization techniques aim to transform queries so that they can be made comparable as a sufficient check for equivalence and to ignore irrelevant syntactic differences when computing edit distances.

One way to test for equivalence could be to use datasets generated by XData. However, it would be very expensive to

*Work partially supported by a fellowship from Tata Consultancy Services
†Currently at Oracle India
‡Currently at Apple India
§Currently at Raymour & Flanigan Furniture and Mattresses

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00159

load each dataset and check for equivalence after each edit. Other techniques for checking equivalence such as Cosette [4] and techniques based on tableau [1], [5], [6] work on a limited set of queries. These techniques also do not provide an efficient way of pruning the search space. In practice, we found that using canonicalization as a sufficient test for equivalence works well. The edit distance computed after canonicalization can also be used as a guidance heuristic for prioritizing edits.

A. Query Canonicalization Rules

In order to reduce irrelevant syntactic differences, we do some initial preprocessing. These steps replace certain SQL operators with other operators, enabling us to reduce the number of types of operators we need to consider during comparison of student and correct queries. For example, `NOT(A>B)` can be replaced by `A<=B`. Some operators like inner joins are associative and commutative. We flatten such operators and construct a flattened tree as shown in Section II-B.

Differences between student and correct queries can also be reduced by semantic canonicalizations. For example, distinct clause on primary keys may be removed and redundant relations in a query may be removed. These canonicalizations can only be applied to queries provided some query and/or database constraints are satisfied. Some of these transformations are widely used in query optimizers to consider alternative query execution plans.

More details on canonicalization rules that we use may be found in [3].

B. Flattened Tree Structure

In order to compare the student query and a correct query, we use a "flattened" tree structure to represent the SQL queries.

For query operators that are commutative and associative such as INNER JOIN, UNION(ALL), INTERSECT(ALL) as well as are predicates involving AND or OR are canonicalized by flattening them. For example $(r \bowtie s) \bowtie t$ is transformed to $\bowtie\ (r, s, t)$. Similarly, for conjuncts of predicates with equality which involve common attributes, the attributes form an equivalence class and the equality conditions may be specified using different attribute combinations. For example, $(A = B) \wedge (B = C) \wedge (C = D)$ can also be specified as $(A = B) \wedge (B = C) \wedge (A = D)$. Regardless of which form is given, the predicate is transformed to $= (A, B, C, D)$.

The flattened tree for a query with a join between 3 relations is shown in Figure 1. Predicates, projections, group by attributes are modeled as special children (connected by a dashed line) may themselves be a subtree. As shown in Figure 1, in case the node is an INNER JOIN this child node would contain all the join and selection conditions. For non-commutative operators like LEFT OUTER JOIN and EXCEPT(ALL) we compare children in order while for commutative operators like INNER JOIN, we ignore the order when matching children.

C. Computing Canonicalized Edit Distance

The instructor can set weights for each query construct. The canonical representations of the student and the correct query

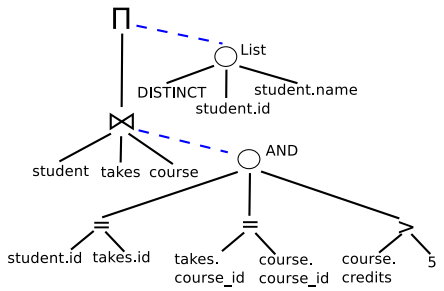

Fig. 1. Flattened Tree

can be used to compute the weighted edit distance between them. We call this edit distance the canonicalized edit distance. The edits could be made by inserting, removing, replacing or moving a node/subtree from one position of the flattened tree to another.

Each query edit has a cost associated with it. The cost of inserting/deleting a subtree within the flattened tree is the sum of the cost of all nodes of the subtree. In the canonicalized flattened tree, each part of the query such as selection, projection, aggregate or subquery is present as a node/subtree. We call each of these parts a component of the query. We find the edit distance for each component separately and then find the weighted edit distance for the query.

The canonicalized edit distance is computed using the formula $\Sigma_{c \in \text{components}} W_c * E_c$ where W_c is the weight of a component and E_c is the edit distance for the component. If the queries are equivalent, the canonicalized edit distance is 0.

D. Using Edit Distance for Partial Marks

One way to award partial marks could have been to deduct marks based on the canonicalized edit distance between the student query and a correct query. However, the partial marks awarded may not be fair because of the following issues.

a) A small edit may significantly reduce the canonicalized edit distance: Consider a correct query to be

```
SELECT * FROM r INNER JOIN s ON (r.A=s.A)
WHERE r.A>10
```

Consider the student query to be

```
SELECT * FROM r INNER JOIN s ON (r.A=s.B)
WHERE s.A>10
```

In the above case, finding the canonicalized edit distance would show two differences: (i) the join condition and (ii) the selection condition. However, if we replace `s.A` in the join condition with `s.B`, in the student query, the queries would become equivalent now. The student query is just one edit away from a correct query, not 2 as the distance above implies.

b) Canonicalizations may increase the edit distance: Consider the case where the correct query to be

```
SELECT DISTINCT id, name
FROM student INNER JOIN takes USING(id)
WHERE takes.semester='Spring'
```

Suppose a student misses the selection condition; the student should be penalized for one error. However, once

canonicalization including redundant join elimination and `DISTINCT` removal is done, the student query becomes

```
SELECT id, name FROM student
```

since the join with `takes` is redundant in the student query and `id` is the primary key of the `student` relation making `DISTINCT` redundant. Now the difference between student and instructor query consists of differences in relations, join operators and join conditions and the distinct operator as well. The canonicalized edit distance is greater than if the query had not been canonicalized. If we first edit the query to add the selection condition, `takes.semester='Spring'` and then canonicalize the query, the problem would not occur.

III. Partial Marking Using Sequence of Edits

We now describe our techniques of partial marking based on the lowest cost edit sequence. Our goal is to edit the student query in order to make it equivalent to a correct query. The minimum number of edits or more precisely the least cost edit sequence gives us a measure of how far the student query was from a correct query; partial marks can be awarded on the basis of the cost of the edits.

An instructor can specify more than one correct query. The techniques described in this section are used to evaluate the student query and award partial marks against each correct query provided by the instructor. The maximum of the partial marks obtained is awarded to the student query.

A. Guided Edits

When editing a student query, potentially an infinite number of edit options are possible. For example, any query predicate can be added as an edit. However, only those edits that make the edited query more similar to the correct query would be useful. In order to narrow down the search space, we edit student queries in a guided manner such that each edit may reduce the difference between the student query and the correct query. Hence, components of the correct query not present in the student query are added to the student query, components of the student query that are not present in the correct query are removed and so on. We call these edits guided edits.

B. Finding Lowest Cost Edit Sequence

There are several possible guided edits for the student query. After an edit is applied to Q to get Q', there may be several more guided edits possible on Q'. A sequence of edits on the student query can make it equivalent to a correct query. Partial marks can be awarded by deducting the sum of cost of edits made on the student query.

Consider a graph whose nodes are all queries for the given schema. For any query Q, edits of the query are also nodes in the graph. Let these edited query be connected to query Q with an edge whose weight is the edit cost. Queries that are canonically equivalent, i.e. their canonical forms are same, are connected by 0 cost edges. The sequence of edits that has the least cumulative cost can now be determined based on the shortest path in this graph from the student query node in the graph to a correct query node. Partial marks can now be awarded based on this shortest path. Since the weight of each edge, which represents the cost of edit is non-negative, the shortest possible path may be found using Dijkstra's shortest path algorithm. Hence, given a set of edits and using a given set of canonicalizations, the shortest path in the graph, as defined above, gives the edit sequence with the least cost. We call the cost of the edits as the *weighted edit sequence distance*.

Theorem 1: In the space of edits considered by our system and in the given space of canonicalization the edit sequence with least cost from the student query to a query that is canonically equivalent to a given correct query can be found using a shortest path algorithm.

C. Shortest Path Algorithm and Greedy Heuristic

To compute the shortest path we take as input a correct query CQ and a student query SQ. The *totalMarks* of the query is computed as based on the number of components in the correct query. We first check for equivalence between the student and correct query using the canonicalization techniques described above. If the queries are not equivalent, we generate edits of SQ and store the edits in a set EQ. When generating edited queries, we reduce the total marks for the edited query by the corresponding edit cost as specified by the instructor.

We now check to see if the query with the highest remaining marks CSQ (this corresponds to the query with the shortest distance from the correct query) in EQ is equivalent to the correct query or not. If the queries are equivalent we have obtained the shortest path and we stop. If the queries are not equivalent, we generate edits of CSQ and add it to EQ. We discard any queries where the total remaining marks is less than 0. The iteration stops when SQ is empty or when a query equivalent to CQ is found. The fraction of marks awarded to the student query is computed by dividing the remaining marks for the edited query by *totalMarks*.

Since the shortest path algorithm considers multiple options at each step, it can be very expensive for queries with a large number of components. We propose, as an alternative, a greedy approach that uses a cost-benefit model. When generating edits of a student query, we consider all guided edits. We also compute the canonicalized edit distance for each edited query as described in Section II-C.

For each edit that is made to the student query, there is some benefit due to the reduction in the canonicalized edit distance. Each edit has a certain cost associated with it as described above. We compute the cost-benefit as *benefit – cost* and use it to pick the best edit for the next step. The remaining queries in EQ are discarded. This helps us prune edits that may not be beneficial; for e.g. removing an extra node from the student query that may have been removed anyway because of canonicalization later.

In practice, when evaluating student queries, we found out greedy technique performs as well as the exhaustive approach in terms of accuracy while taking much less time to run.

IV. Experimental Results

The goal of our experimental evaluation is to find the fairness of the partial marks given by our weighted edit sequence

TABLE I
EVALUATION OF GRADING FAIRNESS

Q. No.	SQ Pairs	CQ	CQ Size	Matches Canon.	Accuracy Canon.(%)	Matches Edit	Accuracy Edit(%)
1	6	1	5	6	100	6	100
2	12	2	9-10	10	83	12	100
3	11	2	11-12	11	100	11	100
4	17	3	13-14	17	100	17	100
5	21	4	13-14	16	76	19	90
6	20	3	14-17	16	80	18	90
7	16	2	18	10	62	15	94
8	25	5	18-25	17	68	20	80
9	21	2	28	10	48	18	86
10	20	3	29	15	75	17	85
11	23	1	35	7	30	23	100
12	6	3	18-37	6	100	6	100
13	30	4	49	9	30	29	97
Total	228	-	-	150	68	211	93

distance using the greedy heuristic. The student SQL queries used in these experiments were taken from submissions in a database course offered at IIT Bombay from 2015 to 2017. The queries used a number of SQL features including subqueries, outer joins, set operators and aggregates with grouping.

Partial marks for previous years were given using different techniques such as canonicalization, manual grading by TA. Also, we were not aware of the grading scheme used by the TAs including which errors in the query were penalized more relative to others. Most importantly, assigning partial marks manually is very difficult, and grades given are only approximate and not necessarily consistent. Hence making a direct comparison between manual partial marking and partial marks generated by our system is not desirable.

Instead, we judged the fairness of our techniques as follows. For each assignment question, we created random pairs of incorrect student queries. We provided these query pairs to two volunteers (without giving them the partial marks awarded using our techniques) and asked them to classify the query pairs into one of the three buckets (a) The first query should get more marks (b) The second query should get more marks (c) Both queries should get almost the same marks although they may have different errors. We then classified the query pairs in the above 3 categories using the partial marks awarded by query edits using the greedy heuristic. If the partial marks differed by less than 10% we classified the query as being almost equal. In Section II-D, we discussed why partial marks awarded based on canonicalized edit distance would not be fair. We also evaluated the effectiveness of partial marks by using the canonicalized edit distance to test the fairness.

The result of the experiment is shown in Table I. The column *SQPairs* shows the number of incorrect student query pairs that we considered. *CQ* shows the number of correct queries used to evaluate the student assignments. *CQSize* shows the number of nodes present in the instructor query and gives some measure of the complexity of the correct queries. *Matches Canon.* indicates the number of student query pairs that were added to the same bucket by our edit sequence based partial marking as well as by the volunteers. *Accuracy Canon.* gives the corresponding accuracy which is computed as *Matched Canon./SQPairs*. Similarly, *Matches Edit* indicates the number of student query pairs that were added to the same bucket by our edit sequence based partial marking as well as by the volunteers, and *Accuracy Edit* gives the accuracy.

While partial marking based on canonicalized edit distance works well for simpler queries, it performs poorly for more complex queries; the overall accuracy is 68%. Our edit based partial marking system works much better and its overall accuracy is 93%. In several cases, we found that a few edits enabled other canonicalizations that made the edited query equivalent to the correct query. Such edits appear to have matched human intuition. For some cases, our canonicalization techniques converted OUTER JOINs to INNER JOINs and removed redundant relations from student queries thus not penalizing their use. The volunteers considered the use of outer joins/additional relations as significant errors. Hence there was a difference in the buckets assigned by volunteers versus our techniques; turning off some canonicalizations would be an option to model human intuition on the degree of error.

We used our system to grade student queries in a database course at IIT Bombay in Autumn 2018. We graded over 1800 student queries automatically, including awarding partial marks to incorrect queries using our techniques. Anecdotally, very few students contested their marks as compared to when they were given partial marks manually.

V. CONCLUSION

The automated grading techniques that we have developed hold great promise for easing the life of instructors, and for helping students learn SQL. The source code binaries available for download from https://www.cse.iitb.ac.in/infolab/xdata. Areas of future work include adding more canonicalization rules like unnesting of subqueries and support for more SQL features such as windowing, ranking and OLAP features. Another area of future work is to cluster similar student queries together to help instructors add appropriate correct answers.

Acknowledgments: We thank Bharath Radhakrishnan for the initial implementation of the canonicalization code. We also thank Ravishankar Karanam and Aarti Sharma for evaluating student queries in the experiment.

REFERENCES

[1] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM J. Comput.*, 8(2):218–246, 1979.
[2] B. Chandra, B. Chawda, B. Kar, K. V. M. Reddy, S. Shah, and S. Sudarshan. Data generation for testing and grading SQL queries. *VLDB J.*, 2015.
[3] B. Chandra, M. Joseph, B. Radhakrishnan, S. Acharya, and S. Sudarshan. Partial marking for automated grading of SQL queries. *PVLDB (Demo)*, 9(13):1541–1544, 2016.
[4] S. Chu, C. Wang, K. Weitz, and A. Cheung. Cosette: An Automated Prover for SQL. In *CIDR*, 2017.
[5] Y. E. Ioannidis and R. Ramakrishnan. Containment of conjunctive queries: Beyond relations as sets. *ACM TODS*, 20(3):288–324, 1995.
[6] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operation. In *VLDB*, pages 535–548, 1978.
[7] 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. In *ICDE*, 2011.