Auto-Marking SQL Assignments at Post Secondary Institution

Michael Liut
University of Toronto - Mississauga
Mississauga, Ontario
michael.liut@utoronto.ca

Ilir Dema
University of Toronto - Mississauga
Mississauga, Ontario
ilir.dema@mail.utoronto.ca

Saihiel Bakshi
University of Toronto - Mississauga
Mississauga, Ontario
saihiel.bakshi@mail.utoronto.ca

Rasagya Monga
University of Toronto - Mississauga
Mississauga, Ontario
rasagya.monga@mail.utoronto.ca

Muyu Wang
University of Toronto - Mississauga
Mississauga, Ontario
sandymuyu.wang@mail.utoronto.ca

Yegor Zadniprovskyy
University of Toronto
Toronto, Ontario
yegor.zadniprovskyy@mail.utoronto.ca

Abstract

To be written by Michael.. in a galaxy far far away...
Insert content here

Keywords

Auto Marking, DDL, DML, SQL, add more...

ACM Reference Format:
Michael Liut, Ilir Dema, Saihiel Bakshi, Rasagya Monga, Muyu Wang, and Yegor Zadniprovskyy. 2020. Auto-Marking SQL Assignments at Post Secondary Institution. In *Proceedings of Auto-Marking SQL Assignments at Post Secondary Institutions (WCCCE 2020).* ACM, New York, NY, USA, 11 pages.

1 Introduction

2 Related Work

2.1 Analysis on Existing Assignment Instructions

We sampled five prominent North American Universities, collecting 15 assignments; seven of them are dedicated for Data Manipulation Language(DML) queries. In this section, our existing assignment question analysis is based on these seven assignments.

2.1.1 Classification of the question

In this part of the analysis, we use Ray V. Rasmussen 's question classification method to analyze the DML assignment question[22]. In Ruasmussen's method, he categorizes questions into four categories; each category is binary, which means the question must fall into either option in the category. These categories are Low-level and High-level language, convergence and divergence, straightforwardness, and structure.

WCCCE 2020, May 8–9, 2020, Vancouver, Canada

Following the results from Part i, find the `playerid`, `namefirst`, `namelast` and `lslg` (Lifetime Slugging Percentage) for the players with the top 10 Lifetime Slugging Percentage. Note that the database only gives batting information broken down by year; you will need to convert to total information across all time (from the earliest date recorded up to the last date recorded) to compute `lslg`.

Order the results by `lslg` descending, and break ties by `playerid` (ascending order).

a) High Level Described Question

Find Winners. Find parties that have won more than 3 times the average number of winning elections of parties of the same country. Report the country name, party name and its party family name along with the total number of elections they have won. For each party included in the answer, report the id and year of the mostly recently won election.

Attribute	Description
countryName	Name of the country
partyName	Name of the party
partyFamily	Name of the family of a party
wonElections	Number of elections the party has won
mostRecentlyWonElectionId	The id of the election that was most recently won by this party
mostRecentlyWonElectionYear	The year of the election that was most recently won by this party
Order by	The name of the country ascending, then the number of won elections ascending, then the name of the party descending.
Everyone?	Include only countries and parties who meet the criteria of this question.
Duplicates?	Countries and party families can be included more than once with different party names.

b) Low Level Described Question

Figure 1: Example of high / low level language described question

2.1.1.1 Low-level and High-level language of Description

What kind of language the assignment uses can affect how students understand the concept of the question. Rasmussen defines a low-level description as one which only requires memorization of role material or simple rephrase the material; and high level description requires some analysis, synthesis, and evaluation are involved[22].

In these seven DML assignments, we define the question that involves analysis, that uses the knowledge or result table from previous questions, or that needs to do extra research of certain terms to write the query as high-level described question; the low-level described question is defined as the question which directly tells the students what they need to do. The example of high-level language described question[14] and low-level language described question[9] are shown in the figure1.

After the analysis, we discovered that most of the questions are described in low language; this indicates that most of the queries are easy to understand and write by students. Also, low-level language described questions make teaching assistants (TA) or automarkers to use less time to mark; most of the low language described questions are binary so TAs can directly lookup the logic of the queries to get the result and the automarker can generate test data

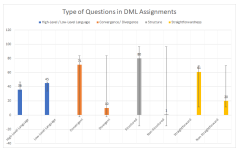


Figure 2: Analysis of Question Types on DML Assignments

to directly compare the result table to give a binary result. This can help universities to reallocate resources to student facing activities, and to make marking the question fairer. For high-level described questions, the automarker needs have the ability to give partial mark based on students' query and result table; however, the automarker needs to have scales to measure students' achievement. The scales of partial marks can be based on students' understanding of the question, which can be reflected in the attributes in the result table or the selectors in query's WHERE clause.

Specifically, we have 61 anonymous submissions for the DML assignment from the University of Toronto, Mississauga campus (UTM). The average grade of high and low-level questions are shown in the graph figure 2. From the statistics, the high-level language described questions have nearly 10% lower average grade than the low-level language described questions. The result shows that the high-level described language could affect students' performance on the question. By analyzing the submission query files, it appears that students may have struggled to understand the question because the students who received little or no partial marks for the high-level language described questions, only wrote the part of sample answer query for the question.

2.1.1.2 Convergence vs. Divergence

Questions also can distribute into convergence and divergence. For convergent questions, it only can lead to one specific answer; but for divergence question, it can lead to multiple answers considered as correct[22].

We analysis the description by details given to student, such as "Delete all recipes written by 'Fred Funk'" [15] is defined as a convergent question in the analysis and "Find the average salary for all physicians in each medical specialty." [16] is a divergent question because it does not tell the students what attributes are showing in the result table; students may have more (or less) columns than the instructor needs in the result table.

We gathered the number of convergent and divergent question descriptions for all of the assignments. In the seven assignments, we have a total of 81 questions, including sub-questions. Most of the questions provide a convergent description to students. Three out of five questions from the University of Toronto, St. George campus (UTSG) DML assignment uses divergent description by words; however, the assignment also uses a table to show the attributes, order, duplicates, and extra requirements for every single

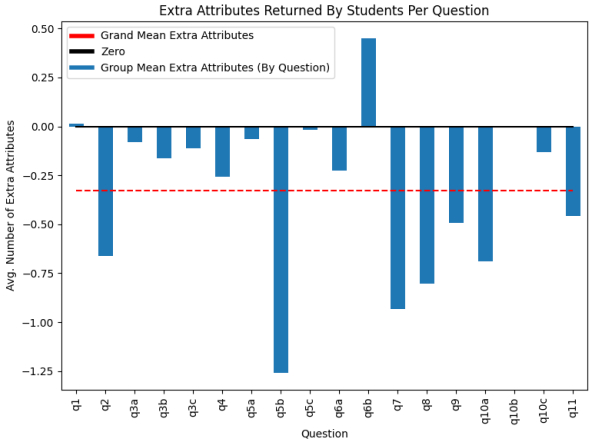


Figure 3: Extra Attributes returned by students per question

question, which makes these three question descriptions from divergent descriptions to convergent descriptions.

As a result, it shows that most of the universities use convergent description for DML query questions(figure 2). That gives a positive sign for the possibility of running the SQL automarker on assignments because convergent description can provide students a clear understanding of the format of the table; that can make automarker easier to compare the result between the students' query and instructor's query. We also need to deal with divergent description questions; for these questions, we can make our automarker less strict – as long as the student provides the result attributes with correct information that the instructor wants, the automarker can provide a setting value for whether the instructor wants to accept more attributes than the standard answer needs.

In the analysis, we static the number of extra rows that the students provided in their answer table for the UTM DML assignment (figure 3). We defined q2,q5b,q4,and q7 as divergent question in the UTM assignment. The figure shows that for all divergent questions, there is a huge misunderstanding for the total number of attributes that the answer needs to have. Especially for q5b, the students are generally missing two rows from the standard answer.

2.1.1.3 Structured vs. Non-Structured

How well the question structured is another problem affect the performance of students' answer and automarker. Structured question gives a specific approach to the students which can help them narrow down the focus and answer the question more quickly [22]; a non-Structured question is the question which takes time to organize a good question [22].

Most of the DDL assignments are structured, because it is easy to tell the correctness by the automarker; automarker can directly run the answer query that the instructor gave and try to balance the score between answer table and students' table. In our collected assignments, all the structured questions can dynamically generate a result table based on test data that either the instructor provided

Figure 4: Type of straightforwardness for a question

or by hypothesis package featured in python.

There is only one non-structured question, which asks "Requires - You figure it out based on both recipes (choose your own amount)" [15]. This is not hard for students to answer but may have a hard time for automarker – the automarker needs to analysis whether students understand the question.

To deal with this type of questions, the automarker should have the ability to substitute the insertion data that the students provided then to compare the final table; or the automarker must compare the fixed value if there is one; for example, in Lawrence case, we can compare whether the student has the essential ingredients.

2.1.1.4 Straight forwardness

A straightforward question is defined as the question itself contains enough information in order to solve the problem. In contrast, a non-straightforward question refers to the question which the students needs to do some searching to gather information for the question [22].

From these seven assignments, most of the questions are straightforward; this gives automarker less workload on analyzing what students want to do because they all get the information they need; however, for non-straightforward questions, they have different types: progressive questions and research needed questions.
First type of the non-straightforward question is progressive question. Progressive questions are the question based on the answer from the previous question(s). In the figure4 a), part b and c of the question are based on part a [16]. When the automarker marks the question, it may treat part b and c unfairly – what if the student did part a wrong? In this case, we need to implement the automarker to have the ability to deduct the mark from part a and may deduct partial marks from part b and c depends on the setting the instructor gives to the automarker; this can allow the automarker fairly treat the result for b and c rather than directly give the students a zero because of the previous part error.

Second type of non-straightforward question is research needed / long information questions; those questions require students to do some research on the background of a question or the question itself contains a long description for the i. For example, figure 2 b) asks students to find how to calculate the histogram[14]; this may affect the performance of the students' result but not a issue for the automarker.

Continue to analyze Figure4, we conclude that straightforward questions have a 7% higher average than the non-straight forward questions. This indicates a positive relationship between the straightforwardness and students' performance on the question. The lowest mark of the non-straightforward questions is a long information question, which has an average of 64.4%. This indicates that students may not want to read a very long description in order to solve the question. Also, it can affect that students may not have a good reading skill. The instructors can provide more long information questions as an additional practise for students; if possible, the university can provide help with training students' reading skill and information understanding skill by holding workshops or after class courses.

2.1.2 Required Operations of the Query

In this section, it talks about how query operation can affect the automarker.

2.1.2.1 Sub-Query

Some of the questions in the seven assignments require the use of sub-queries. This means, if a student makes an error in the sub-query, then the automaker may give a 0 for the question itself. This raises questions – "What does a fair automarker look like?" and "Shouldn't partial marks be awarded to the student?". However, as mentioned before, the down-side of giving partial marks is the cost of human graders. Currently, there are no automarkers that can to take the sub-query out then test it separately and give a partial mark based on each part of the query due to our research.
For this situation, the automaker should have the ability to separate out the sub-query from the main query and run it to see what does it do. If one of the main clause or the sub-query clause is correct, the automarker should give students mark for part of the question. This could help university brings down the cost of running courses and the automarker can give a more precise feedback to the students for the error part. It also could allow us to rearrange the TA hours to office-hour or help session which allows students work with TA in person, then the TAs could have multiple roles which enriches the variety of the helping resources.

2.2 Advantages and challenges of Auto-Marking systems

Assessments in Computer Science courses have been automated to different levels in the past 40 years [7]. The adoption of Auto-Marking systems is mainly motivated by the following:

- Large class sizes require more resources - human (Teaching Assistants) as well as financial and automation can help reduce workload.
- More consistency can be achieved by automating the grading process as chances of human error are reduced [23].
- Feedback can be provided promptly, leading to quicker reflection by the students.
- Accountability of the grading process increases as digital footprints of the Auto-Marking system can help analyze the procedure, whenever required.

While automated grading systems can be advantageous, there are some concerns surrounding their use [23]:

- Partial marks cannot be assigned without significant complexity in system design.
- Assessment criteria needs to be simple enough for automated marking to be consistent.
- Students are required to meet more stringent requirements as there is no human intervention in the Auto-marking process.

With the help of our SQL Auto-Marker (SQAM), we aim to address these concerns and provide a more easily adaptable and robust tool.

2.2.0.1 Auto-Marking programming assignments versus querying languages

There have been numerous attempts to automate grading processes by established institutions. However, the wide variety of courses within the computer science curriculum presents challenges for automation. For example, the methods employed to automate marking of assignments written in Python, Java, or C would be similar but not easily replicable to those involving SQL. The Auto-Marking of assessments specific to programming languages can be achieved by writing suites of unit tests so as to reward marks for passing some criteria. However, Auto-Marking of SQL based assessments requires additional testing such as matching of data in tables which calls for additional complexity in the Auto-Marking system.

A major challenge that impedes the Auto-Marking process is the issue of assigning students partial marks for answers that are not fully correct. This problem of partial marks can be taken care of by using comprehensive unit tests for assignments specific to programming languages like Python or Java. With querying languages however, this issue becomes even more complex. As SQL-based assessments require students to create, query and modify data, mostly in the form of tables in databases, it is challenging to assign marks for partial correctness. This is primarily because there can be several correct ways to query data and give the right output. In addition, there is a possibility that the results generated by the student partially match the expected result, in which case some marks should be rewarded. Therefore, there is a need for a tool that can account for both cases - alternate solutions to the same problem and assign part marks for partially correct results.

2.3 Current Auto-Markers

2.3.0.1 Binary Auto-Marking systems

We examined current Auto-Markers at University of Toronto, St. George, University of Capetown and Napier University to assess the state, compatibility as well as the ability to reward fair partial marks.
PCRS [2] is a web-based assessment application used by University of Toronto, St. George. The system combines interactive programming exercises along with videos for students and has been developed under the supervision of Andrew Petersen. Currently, PCRS supports assessments in Python, C, Java, SQL and Relational Algebra. The SQL based assessments allow students to practice

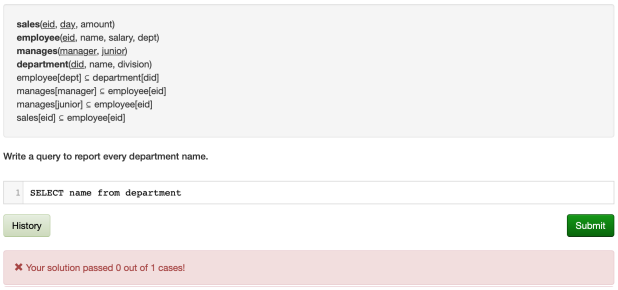


Figure 5: Result of incorrect input in PCRS

writing queries and getting familiar with querying syntax. However, the marking of student submissions on the interface is **binary** in nature. If the submitted query or answer is incorrect, the student gets unlimited attempts for re-submission until the correct query is entered. However, there are no partial marks assigned - the student's submission is either correct or incorrect (Fig. 5)
Although, PCRS offers an effective way for students to practice and refresh their knowledge of data manipulation in SQL, the challenge of partial marks remains since there are none awarded currently.

Another example of a **binary** Auto-Marker is the 'SQL Automatic Marker' developed by the Department of Computer Science at University of Capetown [25]. The application enables course instructors to create and send out assessments along with the ability to add or remove students. The assessments can then be taken by the students on the system which Auto-Marks the students' queries and returns mark summaries.
In order to Auto-Mark given submissions, the system executes the students' queries and checks for correctness. The way the system awards marks depends on either the exact match (or lack thereof) of results from the students' queries or on the presence of certain *keywords* in the submission. These *keywords* are divided into categories A through G and include combinations of basic as well as aggregation clauses like SELECT, FROM, ORDER BY, HAVING and JOIN [25]. Full marks are awarded if the result produced by executing the students' submission is an exact match with the expected result. Whereas, partial marks are awarded if the students' result differs from the expected result but contains keywords from the aforementioned categories A through G. No marks are awarded if the students' result differs and lacks the keywords.

While this system allows for significant reduction in manual work and provides for a better overall assessment experience for the students, the issue of rewarding reasonable partial marks remains. This is due to the numerous possible ways in which the queries could be written. Additionally, certain errors that require to be accounted for include unintended syntactic errors as well as misplaced columns. For example, Fig. 7 shows outputs with misplaced columns.
Therefore, we classified the University of Capetown automarker as **binary**.

Figure 6: Example results of the 'SQL Automatic Marker' (University of Capetown)

CustomerID	CustomerName
4	Around the Horn
11	B's Beverages
16	Consolidated Holdings
19	Eastern Connection
53	North/South
57	Paris spécialités
72	Seven Seas Imports
74	Spécialités du monde

(a) Expected Correct result

CustomerName	CustomerID
Around the Horn	4
B's Beverages	11
Consolidated Holdings	16
Eastern Connection	19
North/South	53
Paris spécialités	57
Seven Seas Imports	72
Spécialités du monde	74

(b) Student result with Misplaced Column

Figure 7: Different order of columns in expected vs student results

2.3.0.2 Research on Partial Marking

Researchers at Indian Institute of Technology, Bombay are developing a platform called *XData* for automated grading [3]. In order to tackle the issue of partial marking, *XData* develops several datasets pertaining to a particular query. First, the submitted query as well as the instructor's expected queries are 'canonicalized' to make the format more standard and are then broken into several components [3]. Due to the possibility of syntactic variation in SQL queries (as they can produce the same output in different ways), the researchers allow for alternative solutions to be uploaded by the instructor. The system, thus compares the submitted query to each of these alternative queries given by the instructor, designates partial marks based on those comparisons and then finally computes the maximum of such marks achieved.
In the process of 'canonicalization' performed by *XData*, equivalence classes of certain attributes are also taken into account [3]. For example, due to the presence of the *JOIN* condition in the query:

```
SELECT employee.deptId FROM  employee INNER JOIN
department ON employee.deptId=department.id
```

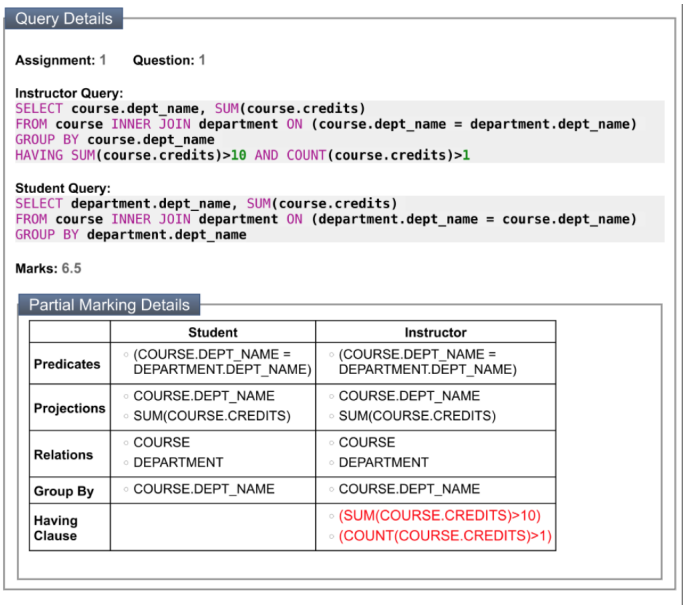


Figure 8: Sample results from *XData*

Here, 'SELECT employee.deptId' could be replaced with 'SELECT department.id' as they give the same output [3]. Therefore, a change in an attribute belonging to a particular *equivalence class* with another attribute of the same class would not lead to a change in the overall query result, as long as the change occurs above the *JOIN* condition(s). The *XData* system replaces occurrences of an attribute above the *JOIN* condition, by the lexicographically least variable within its equivalence class [3].
Another important component of the 'canonicalization' mechanism in *XData* is to check for functional dependencies [3]. The system 'canonicalizes' *ORDER BY* clauses by removal of some attributes that can be functionally determined by others appearing before the *ORDER BY* clause [3].
Finally, comparisons are made between the queries derived from the 'canonicalization' steps. This is done through a technique that matches syntactical components and assigns weights accordingly [3].
Upon completion, *XData* enables students to see a component wise breakdown of their query in comparison to the instructor's query along with the mark received (Fig. 8)

2.3.0.3 Refined Auto-marking Systems

Another attempt at Auto-marking SQL assignments has been successfully made by researchers at Napier University [8]. The main focus of the examiners was to create an online learning environment for students that could essentially replace a conventional tutorial experience.
In the learning environment created by Napier University, the system analyses the submitted query and displays the results. This analysis is shown via a *SQL Accuracy* score that assigns a percentage grade between 0 to 100 [8]. To give the students a clear idea of the partial correctness of their submitted query, a *row/column*

NAME	POPULATION	REGION
Brazil	172860370	South America
China	1261832482	Asia
India	1014002817	Asia
Indonesia	224784210	Southeast Asia
United States	275562673	North America

Figure 9: Partially correct (highlighted) output

analysis is also provided with those parts of the table highlighted which were considered correct [8].
For example, this query would be assigned a 100% correctness score:

```
SELECT name, population
FROM cia
WHERE population>200000000
```

However, the the following query would be given a 56% correctness score:

```
SELECT name,population,region
FROM cia
WHERE population>100000000
```

The correctness scores for the two queries differ by 44% because of a missing column as well as a difference in values of the 'population' field. This is certainly a sizeable difference between the two scores and may or may not be seen as a fair assessment of the student's submission. It could be possible that the student failed to correctly interpret the question or rather just made a typographical error. Thus, the justification of this difference in scores would depend on the instructor(s).
Along with analyzing partial correctness, the system provides a table with highlighted sections denoting correct output (Fig. 9 [8]). Furthermore, this online learning environment made by researchers at Napier University assesses queries according to string similarity between students' submissions and instructors' solutions [8].

2.4 Comparison to SQL Auto-Marker (SQAM)

One of our primary goals while building SQL Auto-Marker has been to incorporate universal design in order to reduce the number of requirements needed from the instructors' end. This can allow for easier adoption and use of the Auto-Marker by instructors from other universities as well.

2.4.0.1 Compatibility of other Auto-Markers

The other Auto-Markers that we have examined in the previous subsection from University of Toronto, St. George, University of Capetown as well as Napier University, require instructors to create assessments on a specific platform. The students are then allowed to take assessments on the same platform before Auto-Marking is done. However, our approach with the **SQL Auto-Marker (SQAM)** has been to minimize dependencies and requirements so as to allow for more robust and universal adoption of the Auto-Marker. Additionally, our Auto-Marker aims to provide constructive feedback to not just students (based on their submissions) but also to instructors so they can assess and improve assignments in an iterative manner.

With the current **SQL Auto-Marker (SQAM)** architecture in place along with requirements, instructors from different universities will be able to accomplish Auto-Marking without drastically changing the way their current assessments exist.

3 Methodology and Theory

3.1 Partial Marks

As in most Computer Science assignments, a solution to a problem can be one element in a large finite set of solutions, and commonly, as the complexity of the problem increases the number of acceptable solutions tends to grow too. In order to combat this, we move away from awarding students grades based on their actual SQL query, but rather based on the results generated by it. However, students' solutions can also have small mistakes or typo's that do not cause syntax errors but instead produce slightly different results. For instance, the problem of misplaced columns discussed above. Assigning a binary grade to a student's answer can, as a result, lead to them scoring a significantly lower grade than what their solution might deserve. This is because it promotes fairness in the grading scheme, for instance a student with a small error should not be given the same grade as a student who did not even submit an answer.

Our proposed auto-marking solution deals with the aforementioned issue of partial marking by assigning student's queries a part mark based on the similarity of their results to that of the instructor's solution. Additionally, in order to deal with problems such as misplaced columns, or rows returned in a certain sorted order, we sort both the instructor's solution and the student's solution before estimating the similarity between them. The exception to sorting the output is when the order of the results is of importance to the question. In this case, the professor has the option to evaluate the results without sorting.

3.1.1 Test Cases

In order to assign partial marks to a student, we generate test cases for each DML query based on the mark breakdown provided by the instructor. For instance, if a query is worth 5 marks of the assignment, we generate 5 Python unit test cases (20% similarity, 40% similarity, 60% similarity, 80% similarity, 100% similarity) to assess the similarity of the two solutions. If a student's solution was 65% similar to the solution, the student passes the first 3 test cases and is awarded a grade of 3/5 on the question. Additionally, the instructor is given the choice to set the level of rounding that the auto-marker conducts with a default set to *Gaussian* rounding. Otherwise, the rounding setting can be turned off completely.

Moreover, the usage of test cases provides the student with more granular feedback. This allows them to evaluate sections where they made mistakes, and how these mistakes can be rectified.

3.1.2 String similarity metrics

In order to best measure the correctness of a student's query results we employ four different string similarity metrics. We compare the

sorted results of the student's query against that of the instructor's solution. Since different similarity metrics are best suited for different types of strings, we use all the four metrics on each set of results and utilize maximum average similarity to grade the student's query. The four metrics used are discussed below.

1. Levenshtein Distance:
Given two strings A and B, the *Levenshtein* distance is the edit distance between A and B. Edit distance is the minimum number of edit operations required to transform string A into B. These operations are defined as the insertion of a symbol, the substitution of a symbol for another, and the deletion of a symbol [19]. Using the *Levenshtein* distance, the similarity ratio is calculated as:

$$\frac{(|A| + |B|) - lev(A, B)}{|A| + |B|}$$

where $|A|$ and $|B|$ are the lengths of string A and B respectively, and $lev(A, B)$ is the *Levenshtein* distance from A to B. Note, the *Levenshtein* distance takes into consideration the case of the strings as well, so "Auto Marking" and "auto marking" have a *Levenshtein* distance of 2.

2. Jaro-Winkler Distance:
The *Jaro-Winkler* distance is another edit distance metric that uses a prefix scale, and as result produces higher similarity scores for strings with identical prefixes. The *Jaro-Winkler* distance is an improvement on the *Jaro* distance, which considers the number of matching symbols in two strings as well as the number transpositions between them [6]. Winkler's improvement gives higher similarity scores to strings that start with the same symbols. [6]

3. FuzzyWuzzy: Sorted Token Similarity:
Token Sort ratio is a string similarity measure that converts the strings into tokens and sorts the tokens alphabetically before calculating the similarity measure [5]. This distance measure is implemented in the *Fuzzy Wuzzy* Python package created by "SeatGeek" [5]. The advantage of this algorithm is that it sorts words within the string of results before computing the similarity [5].

4. Ratcliff-Obershelp Similarity:
Ratcliff-Obershelp is a sequence based string similarity algorithm that computes the longest common sub-string from the two strings and splits the string at this point, and then recursively repeats this process on the split strings [10]. As a result, the sequence of strings are split into smaller common chunks. The similarity score is given by the ratio of twice the number of common characters to the total number of characters in the strings.

The first two of four of the similarity metrics are edit-distance based metrics and hence equal weight is given to every character of the strings. Whereas, in a token based similarity sequence of tokens of varying length have equal weights. Finally, in the sequence based similarity algorithm equal weight is given to equi-length combinations of characters.

	Word A	Word B	Levenshtein	Jaro-Winkler	Fuzzy
0	Person	person	0.833333	0.888889	
1	chien	niche	0.600000	0.600000	
2	Emily	Jack	0.000000	0.000000	
3	122009	1232009	0.923077	0.961905	
4	Emily, Jack, John	Jack, Emily	0.060606	0.730838	
5	New Hampshire	New York	0.576923	0.669872	

Figure 10

3.1.3 Implementation details

We store each row returned from the relational database as a tuple of strings and the table as a list of tuples. In order to measure the string similarity between each row of a student's result with that of the solution, we convert each tuple of strings to a list of strings and recursively sort each list. Simply, we first sort each list of strings corresponding to a row of the table, then we sort the list of lists corresponding to all rows of the table. However, for questions where the order of the rows is required, the instructor has the option to toggle the sorting of the rows off. In this case the rows of both the student's and instructor's results are not sorted for the specified query.

3.2 Clustering submissions

In this section, we explore the possible approaches to clustering student submissions. We tested our approach for the assignment that contains submissions of 51 students, each submission contains student's solutions (sequence of SQL instructions) to 7 tasks. We used scikit-learn to implement all stages of the preprocessing, vectorization and clustering [21].

3.2.1 Motivation

Providing personalized feedback is an important part of the educational experience. However, due to an open-ended nature of the problems in SQL assignments providing personalized feedback requires manual annotation of all submissions, which is, most of the time, impossible. Finding a way to group submissions with similar approaches should help to minimize the effort of teaching staff needed to provide personalized feedback. The ability to see clusters in the solution space may also be helpful for the course instructors since it should allow them to have a better understanding of the state of their class [11].

3.2.2 Feature Representation

We vectorized the solutions using the Bag of Words model. The BoW model assigns a unique index to each word. Then any document can be encoded as a fixed-length vector with the length of the vocabulary. The degree to which the corresponding word is present in the document determines the value of each position of

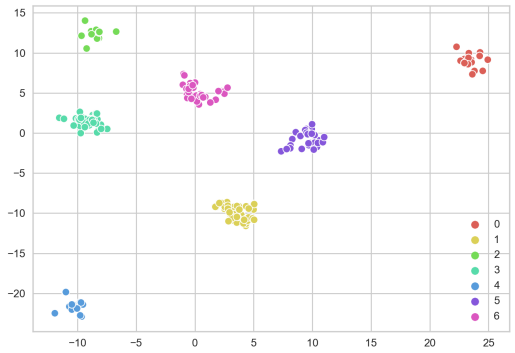


Figure 11: PCA (0.95 variance explained) + t-SNE

the vector. In our case, we filled the vector with a count of each word in the encoded document [13] [4].

Using more advanced BoW encoding techniques could lead to "better" embeddings (e.g., term frequency–inverse document). However, other approaches tend to be more sensitive to noise in data.

When vectorized using BoW model, each query will typically be a very sparsely populated vector living in a very high-dimensional space. This problem, however, can be easily solved by applying standard dimensionality-reduction algorithm such as Principal Component Analysis [17]. Our assignment consists of $51 \times 7 = 357$ vectors and uses the vocabulary size of more than 165 words/tokens. The first 25 components (corresponding to 25 highest eigenvalues of the covariance matrix) roughly hold around 95% of the total variation in the data. Thus, we are able to significantly reduce the dimensionality of the data giving up only little variation.

In Figure 11 we used T-Distributed Stochastic Neighbouring Entities (t-SNE) to visualize our dataset after the dimensionality reduction using PCA [24]. Each point corresponds to a student's solution and the point's colour correspond to the task that it belongs to. We observe very clear separation between the solutions to different tasks.

3.2.3 *Clustering methods*

Armed with a vectorized version of students' solution we can now proceed to clustering. Our goal is to find meaningful clusters in the solution space.

We used K-Means and Gaussian Mixture Model (GMM) techniques for clustering. The results of clustering for each model is presented using a "number of students" × "number of tasks" matrix where each column of the matrix corresponds to the true section the query belongs to, each row corresponds to submission of a single student, entries of the matrices are coloured according to results of the clustering (different colours - different clusters). This is simply a convenient format to analyze model's performance on the dataset.

We used the 2-dimensional version of our dataset (the embeddings from Figure 11) to illustrate how, as we increase the number of components in GMM, the solutions that belong to the same task

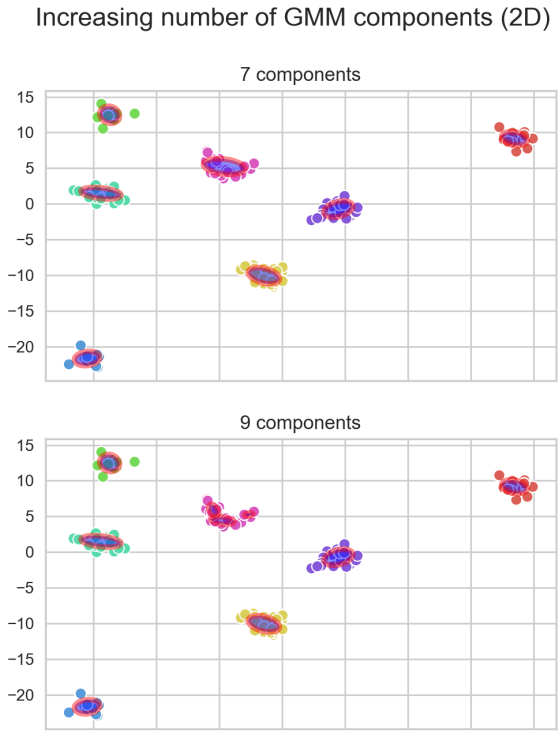


Figure 12: GMM clusters visualization after training on the 2D version of the original dataset (BoW + PCA + t-SNE). The number of components increases from 7 to 9.

are being partitioned into groups based on their syntactic similarity. We visualized the clusters for GMM trained with 7 (the number of tasks), 9, 11, and 13 components. It's important to remember that t-SNE can often produce misleading results, so we only use it for visualization purposes.

Figure 14 and Figure 15 illustrate how the clustering results change as we increase the number of clusters in our model from 7 to 13 (from left to right) and train it on the embeddings we derived by performing PCA on BoW vectors. Again, observe that new clusters lie almost entirely within one task (one column). The clusters within a single task can be thought of as different approaches to the same task. So spotting a common approach becomes possible just by investigating a single plot. This kind of feedback provides instructor with useful tools to analyze the class performance and partially automate the process of providing personalized feedback.

One of the heuristics to help the instructors choose the number of clusters is the "elbow" method. The sum of distances of samples to their closest cluster centre is typically used as a scoring metric for K-Means and the Silhouette Score is used as a scoring metric for GMM. In Figure 16 one of the possible number of clusters which we can pick is 21, since there is not much gain in increasing the number of clusters afterwards. An average of 3 clusters per task seems like a reasonable number of possible approaches for the given problems.

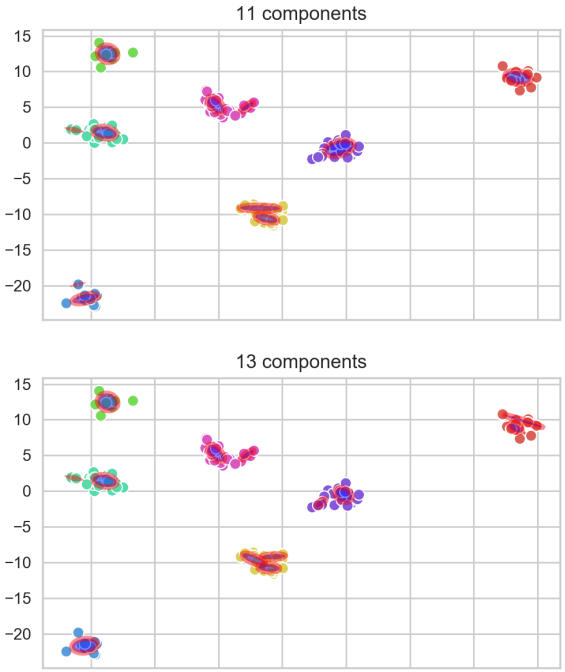


Figure 13: GMM clusters visualization after training on the 2D version of the original dataset (BoW + PCA + t-SNE). The number of components increases from 11 to 13.

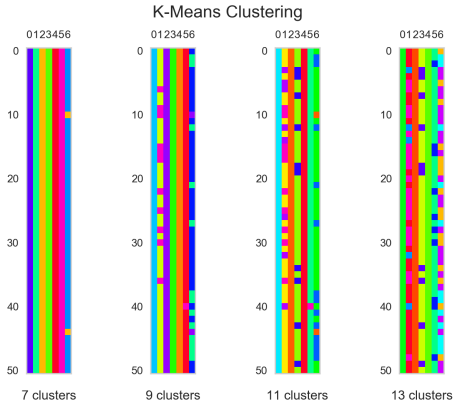


Figure 14: PCA (0.95 variance explained) + K-means

3.2.4 Future work

4 The Application

4.1 SQAM architecture overview

In this section we give an overview of the SQAM architecture with its components and an outline of the data flow. A brief description of SQAM components is given below:

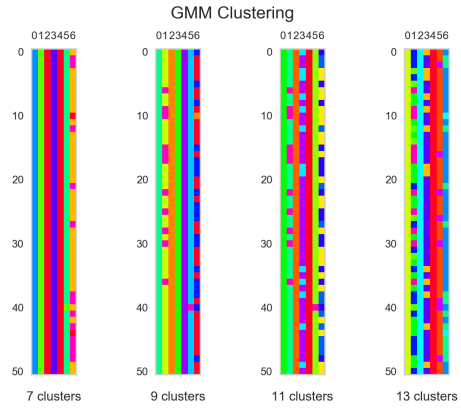


Figure 15: PCA (0.95 variance explained) + GMM

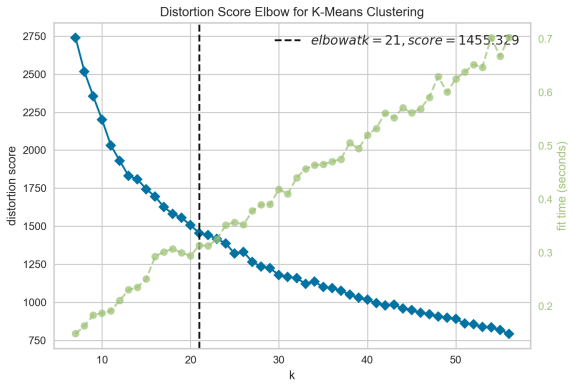
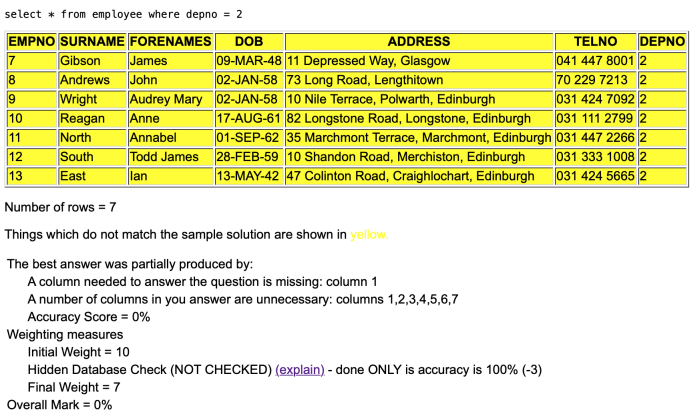


Figure 16: "Elbow" curve for K-Means

- **Submission Spider** parses the class submission directory and returns a mapping from groups to the list of paths to the corresponding submission.
- **Query Extractor** prepossesses and parses the content of each group's submission and for each group generates a mapping from sections of assignment to corresponding queries.
- **Query Runner** performs the database setup, runs the queries and performs all the necessary database-related maintenance and checks. For each group **Query Runner** returns a mapping from sections of assignment to the query result.
- **Grader** calculates the results for a group's submission given expected and actual outputs and the given heuristic.
- **Aggregator** aggregates all the group's result and calculates descriptive statistics based on the results of the whole class.

An essential part of grading any type of assignment is to provide detailed and rapid feedback so that students can improve accordingly. Student learning and feedback from instructors should be conjointly maintained otherwise the assessment's formative aspect is lost [20].



```
select * from employee where depno = 2
```

EMPNO	SURNAME	FORENAMES	DOB	ADDRESS	TELNO	DEPNO
7	Gibson	James	09-MAR-48	11 Depressed Way, Glasgow	041 447 8001	2
8	Andrews	John	02-JAN-58	73 Long Road, Lengthitown	70 229 7213	2
9	Wright	Audrey Mary	02-JAN-58	10 Nile Terrace, Polwarth, Edinburgh	031 424 7092	2
10	Reagan	Anne	17-AUG-61	82 Longstone Road, Longstone, Edinburgh	031 111 2799	2
11	North	Annabel	01-SEP-62	35 Marchmont Terrace, Marchmont, Edinburgh	031 447 2266	2
12	South	Todd James	28-FEB-59	10 Shandon Road, Merchiston, Edinburgh	031 333 1008	2
13	East	Ian	13-MAY-42	47 Colinton Road, Craighlochart, Edinburgh	031 424 5665	2

Number of rows = 7

Things which do not match the sample solution are shown in yellow.

The best answer was partially produced by:
A column needed to answer the question is missing: column 1
A number of columns in you answer are unnecessary: columns 1,2,3,4,5,6,7
Accuracy Score = 0%
Weighting measures
Initial Weight = 10
Hidden Database Check (NOT CHECKED) (explain) - done ONLY is accuracy is 100% (-3)
Final Weight = 7
Overall Mark = 0%

Figure 17: All rows and columns are highlighted due to no matches

4.1.0.1 Annotated feedback provided by Napier University

The Auto-Marker developed by instructors at Napier University [8] provides feedback in the following ways:

- A table of results that highlights (in yellow, Fig. 17) parts of the student's query that do not match with the expected output.
- Details of columns that are missing or unnecessary in the submitted query. An *accuracy score* is assigned based on the similarity of contents in the student's output versus the expected output.
- Lastly, a *hidden database check* is performed when 100% *accuracy* is achieved in order to estimate the robustness of the submitted query against other data (This feature is still under review by the researcher) [8].

4.1.0.2 Annotated feedback provided by SQL Auto-Marker (SQAM)

The SQL Auto-Marker (SQAM) in contrast, provides feedback differently as it does not require instructors to upload assessments on a web-based platform. SQAM creates a *report.txt* file in each of the group submissions' folders once Auto-Marking is complete. In addition, a more general *report.txt* file is created in the instructors' directory that displays overall results of all group submissions.
The *report.txt* file in the group submissions' directory shows how close the student's submission is from the sample solution provided by the instructor. This is a correctness percentage that depends on the maximum marks possible for a specific question.(Fig. 18)
If there exists a difference between tables created from submitted solutions and sample solutions, then a comparison is shown between columns *expected* and columns *provided* along with any missing rows from the students' table.
Moreover, if any extra data exists in the students' table, then it is displayed in order for the student to better understand their query result. (Fig. 19)

```
Summary of Results: 72 out of 80 tests successfully passed
--

Tests for q1 (3/3 passed)
--

1) Query: q1 -- Answer is 33.33% correct.                                    .. ok!
2) Query: q1 -- Answer is 66.66% correct.                                    .. ok!
3) Query: q1 -- Answer is 100.00% correct.                                   .. ok!


Tests for q2 (2/4 passed)
--

1) Query: q2 -- Answer is 25.00% correct.                                    .. ok!
2) Query: q2 -- Answer is 50.00% correct.                                    .. ok!
3) Query: q2 -- Answer is not 75.00% correct.                                .. failed
.. because Results produced by query q2 are 58.88% different from correct solutions. (details below):

    Details below...
```

Figure 18: SQAM feedback per question

```
    Columns Required:
        ('DrugCode', 'Name', 'TotalSales')
    Columns Provided:
        ('DrugCode', 'Name')
    Note: If provided columns are correct but named differently, marks were not deducted.
    Rows missing from student results:
        {('18400', 'Acetaminophen (Liquid)', 'drg_28850201')}
    Extra rows in student results:
        {('Acetaminophen (Liquid)', 'drg_28850201')}
```

Figure 19: SQAM feedback - extra rows in student output

5 Testing and integration with UAM

In this section we provide testing scenarios and describe integration of our proposed system with the UAM (University of Toronto Universal Automarker).

The modular architecture of SQAM enables the use of well-known software testing techniques, such as black-box (functional) and white-box (code structure) testing[1], on its major components. In addition of testing each component in isolation, we have also conducted integration testing, code reviews, and analyzed complexity metrics of it.

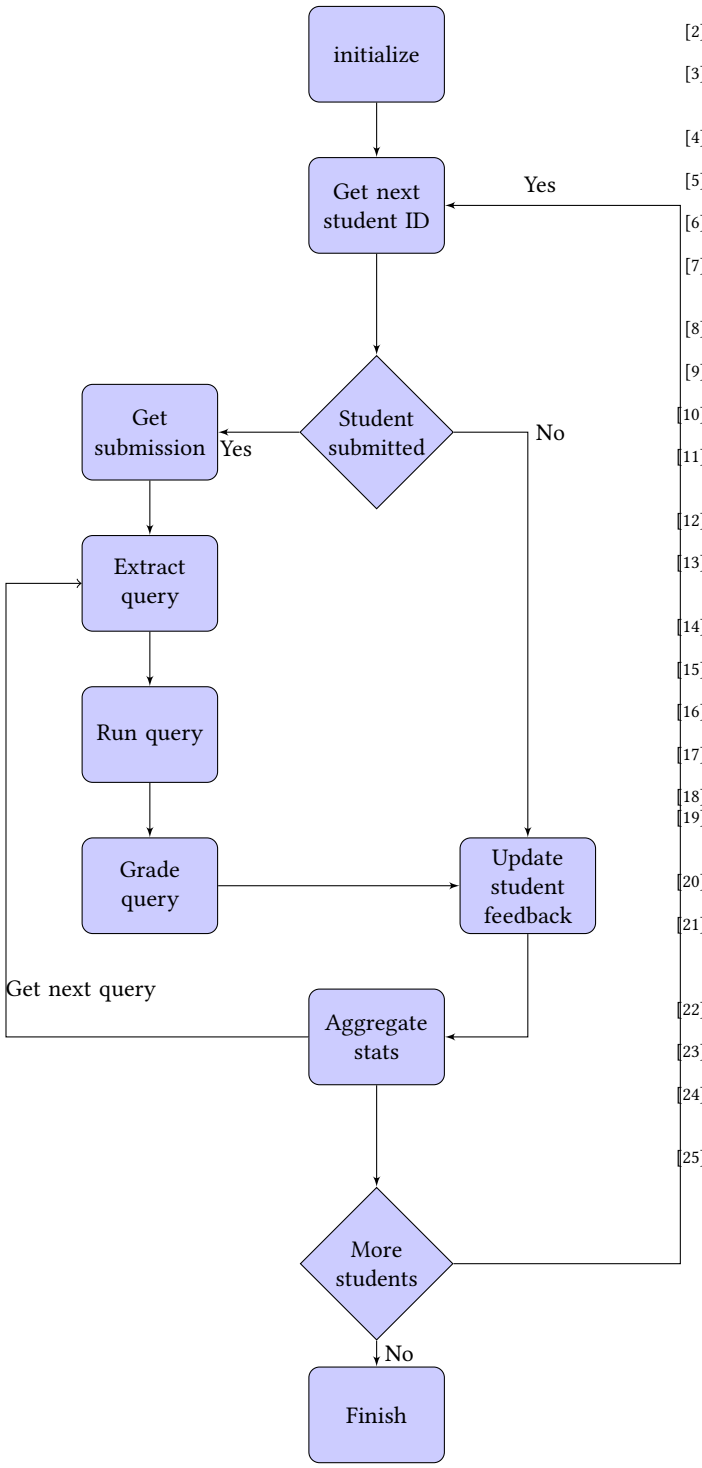
The input to the automarker consists of a set of student's submissions, each composed of various queries. A fundamental requirement of back-box (functional) testing is classification of the input into equivalency classes, generation of the equivalency classes, and boundary cases for each pair of neighboring equivalency classes [12].

A basic requirement of input classification into equivalency classes, is the equivalency classes must be non-intersecting.

First, we consider the black box case. Since student's solutions for similar requirements differ based on the assignment type, a basic input classification can be obtained by taking the cross product types of assignments with the following groups of submissions, based on the outcome of the testing process: correct submissions, partially correct (further subdivided based on 3.1.1), and incorrect, which may further classify as syntactically correct, but outcome is completely different from the expected outcome, and syntactically incorrect. Some important boundary cases to include would be cases when there is no submission at all, empty submission, or submission that contains invalid SQL statements.

Next, we consider the white-box case. Here the input is classified based on the traversal of the flow graph[18], therefore the number of equivalency classes is approximately equal to the cyclomatic complexity of our implementation.

For both cases, we have worked out test cases based on the following chart, that represents the flow of the submissions through SQAM.



References

[1] Sarika Chaudhary Akanksha Verma, Amita Khatana. 2017. A Comparative Study of Black Box Testing and White Box Testing. *International Journal of Computer Sciences and Engineering* 5 (2017). Issue 12.

[2] Diane Horton Andrew Peterson. 2018. *SQL and Relational Algebra Exercises in PCRS.* https://mcs.utm.utoronto.ca/~pcrs/sql-programming/index.shtml

[3] Bharath Radhakrishnan Shreevidhya Acharya S. Sudharshan Bikash Chandra, Mathew Joseph. 2016. Partial Marking for Automated Grading of SQL Queries. (2016). https://doi.org/10.14778/3007263.3007304

[4] J Brownlee. 2017. How to Prepare Text Data for Machine Learning with scikit-learn.

[5] Adam Cohen. [n.d.]. FuzzyWuzzy: Fuzzy String Matching in Python. https://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/

[6] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. 2003. A comparison of string DISTANCE metrics for name-matching tasks. *IIWeb* 2003 (05 2003).

[7] Livingstone D. Douce, C. and J. Orwell. 2005. Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing* 5 (2005). Issue 3.

[8] Andrew Cumming Dr. Gordon Russell. 2004. Improving the Student Learning Experience for SQL using Automatic Marking. *CELDA* (2004).

[9] Diane Horton. 2019. *DML assignment for CSC343H1 at the University of Toronto.* http://www.cs.toronto.edu/~dianeh/

[10] Ratcliff J and Metzener D. 1998. Ratcliff-Obershelp PatternRecognition. *Black P (ed.), NIST* Dictionary of Algorithms and Data Structures (1998).

[11] Andy Nguyen Jonathan Huang, Chris Piech and Leonidas Guibas. 2013. Syntactic and functional variability of a million code submissions in a machine learning MOOC. *AIED Workshops Proceedings* 25 (2013).

[12] Paul Jorgensen. 2014. *Software Testing: A Craftsman's Approach, Fourth Edition.* CRC Press, Boca Raton, FL.

[13] Dan Jurafsky and James H. Martin. 2009. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* Pearson Prentice Hall, Upper Saddle River, N.J.

[14] David D. Kim. 2020. *DML assignment for CS186 at the University of California, Berkeley.* https://mcs.utm.utoronto.ca/~pcrs/sql-programming/index.shtml

[15] Ramon Lawrence. 2020. *DML assignment for COSC304 at the University of British Columbia, Okanagan.* https://people.ok.ubc.ca/rlawrenc/

[16] Michael Liut. 2019. *DML assignment for CSC343H5 at the University of Toronto.* https://www.michaelliut.ca/

[17] Benjamin Fayyazuddin Ljungberg. 2017. *Dimensionality reduction for bag-of-words models: PCA vs LSA.* Master's thesis.

[18] Thomas J. MacCabe. 1983. *Structured Testing.* IEEE Computer Society Press.

[19] A. Marzal and E. Vidal. 1993. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15, 9 (1993), 926–932.

[20] K. Reiling P. Orsmond, S. Merry. 2000. The use of student derived marking criteria in peer and self-assessment. *Assessment Evaluation in Higher Education* (2000).

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[22] Ray V. Rasmussen. 1984. Practical Discussion Techniques for Instructors. *Alberta Association for Continuing Education* (1984).

[23] Hussein Suleman. 2008. Automatic marking with Sakai. 338 (01 2008), 229–236. https://doi.org/10.1145/1456659.1456686

[24] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605. http://www.jmlr.org/papers/v9/vandermaaten08a.html

[25] Rian Tshepe Zach Meltzer, Elvis Sebatane. 2019. CSC3003S Capstone Project: SQL Automatic Marker. (2019).