

Fall 2023

EvalSQL - AUTOMATED ASSESSMENT OF DATABASE QUERIES

Damanpreet Kaur

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects



Part of the [Other Computer Engineering Commons](#)

Recommended Citation

Kaur, Damanpreet, "EvalSQL - AUTOMATED ASSESSMENT OF DATABASE QUERIES" (2023). *Master's Projects*. 1314.

DOI: <https://doi.org/10.31979/etd.bs5q-k5u4>

https://scholarworks.sjsu.edu/etd_projects/1314

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

EvalSQL - AUTOMATED ASSESSMENT OF DATABASE QUERIES

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Fulfillment

of the Requirements for the Degree

Master of Science

by

Damanpreet Kaur

Dec 2023

© 2023

Damanpreet Kaur

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Project Titled

EvalSQL - AUTOMATED ASSESSMENT OF DATABASE QUERIES

by

Damanpreet Kaur

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

Dec 2023

Dr. Chris Pollett Department of Computer Science

Dr. Ben Reed Department of Computer Engineering

Dr. Jahan Ghofraniha Department of Computer Science

Dr. Fabio Di Troia Department of Computer Science

ABSTRACT

EvalSQL - AUTOMATED ASSESSMENT OF DATABASE QUERIES

by Damanpreet Kaur

In computer science programs, database is a fundamental subject taught through several undergraduate courses. These courses develop theoretical and practical concepts of databases. Building queries is a key aspect of this learning process, and students are assessed through assignments and quizzes. However, grading these assignments can be time-consuming for professors, and students usually receive feedback only after the deadlines have passed. As a result, students may miss the opportunity to improve their work and achieve better grades. To address this issue, it would be beneficial to provide students with immediate feedback on their submissions. EvalSQL is an automated system that allows the evaluation of assignments to provide constructive feedback to students on Canvas after submissions. The feedback is based on the correctness of the query, the state of the database after a query execution, and keyword matching. This would allow them to identify their mistakes and correct them promptly, which can lead to a better learning experience and improved grades. Additionally, professors could benefit from a streamlined evaluation process that allows them to focus on teaching and other tasks.

Keywords- Database systems, SQL, Automated evaluation, Immediate feedback

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my thesis advisor Dr. Ben Reed and Dr. Chris Pollett, for their invaluable guidance, support, and encouragement throughout my research project. Their vast knowledge and expertise in the field of Software development have been instrumental in shaping my ideas and refining my approach to the study.

I am also deeply grateful to the faculty and staff of the Computer Science at San Jose State University for providing me with an excellent academic environment and the resources needed to carry out my research. Special thanks go to Dr. Fabio Di Troia and Dr. Jahan Ghofraniha for their insightful feedback and helpful suggestions.

My sincere appreciation goes to my husband Keshav Sharma, who has been a constant source of motivation and inspiration and has supported me in every step of the journey.

Thank you all for your support and encouragement throughout this journey.

TABLE OF CONTENTS

CHAPTER

1	Introduction	1
2	History and Background	3
2.1	SQL Automatic evaluation/ grading systems	4
2.2	SQL Error Categorization	8
3	System Design and Methodology	10
3.1	Objective of EvalSQL	10
3.2	System Inputs	11
3.3	MySQL-based query execution on Docker containers	12
3.4	High Level Design	13
3.5	Processing SQL queries	13
3.5.1	Processing DDL statements	14
3.5.2	Processing DML statements	15
3.5.3	Processing DQL statements	16
4	Survey	18
4.1	Pre-execution survey	18
4.2	Post-execution survey	19
5	Implementation	22
5.1	CodEval	22
5.1.1	The specification file	22
5.1.2	The Docker and the system	22

5.2	The Docker image updates	25
5.3	Specification file updates	26
6	EvalSQL Execution	32
6.1	Setup	32
6.2	Specification file creation	32
6.3	Running EvalSQL	34
6.4	Evaluating submissions	35
7	Difficulties and Challenges	38
7.1	Building MySQL server and client in one Docker container	38
7.2	Execute multiple scripts on docker container start	38
7.3	MySQL user permissions to access LOAD statement files	39
7.4	Individual files for SQL statements	39
8	Limitations and Future Works	40
8.1	Extending support to other database systems	40
8.2	Adding JDBC support	40
8.3	Adding scores to evaluation	41
8.4	Displaying detailed feedback outputs	41
8.5	Breaking down the queries to identify logic build	42
9	Conclusion	43
	LIST OF REFERENCES	44
	APPENDIX	

LIST OF TABLES

1	Describes the positive aspects considered in the above figure. [1] .	4
2	Comparison for different tools.	9
3	Votes count in each category.	19
4	Specification tags in CodEval. [2]	23
5	Specification tags in EvalSQL.	28
6	List of SQL constraints and associated keywords in SCHEMACHECK tag.	31

LIST OF FIGURES

1	Positive aspect of the automated assessment tool. [1]	3
2	TestSQL User Interface. [3]	5
3	System Design for evaluating SQL [4]	7
4	Most common error frequencies for all queries.	8
5	Most common error frequencies for multi-table queries.	9
6	System design with Canvas, CodEval, and EvalSQL.	14
7	Sample Database Entity relationship diagram for assignment. . .	15
8	Survey results of the automated assessment tool.	18
9	Survey question 1 of the automated assessment tool.	20
10	Survey question 2 of the automated assessment tool.	21
11	Survey question 3 of the automated assessment tool.	21
12	Specification file for CodEval. [2]	24
13	Dockerfile for EvalSQL container.	27
14	TSQL tag implementations.	29
15	TSELECT tag implementations.	29
16	SCHEMACHECK and CONDITIONPRESENT tag implementations.	30
17	Configuration file for EvalSQL.	32
18	Sample Specification file for EvalSQL.	33
19	Specification file Uploaded for EvalSQL.	34
20	Sample Scheduling Jobs for EvalSQL.	35

21	Failed execution for EvalSQL.	36
22	Passed execution for EvalSQL.	37

CHAPTER 1

Introduction

Problem-based learning is a teaching approach that stimulates students' learning by presenting them with a real-life problem that requires a solution [5]. It has been practiced in teaching and evaluating computer science courses as well [6]. Databases and Database Management Systems (DBMS) have been an integral part of the computer science curriculum. It is a foundation-level course taught as part of a Computer Science and Software Engineering degree program. The application of relational databases is being taught using Structured Query Language (SQL). SQL defines the paradigms for designing queries to create, access, and retrieve data from databases. Learning and implementing SQL queries is important for students to excel in database courses [7, 8]. The SQL query implementation is evaluated using assignments and lab exercises in such courses. Students build SQL queries from natural language input on a given database. Mastering query designing skills and converting a given plain-text scenario to an SQL query requires many trials and errors to achieve the required solution [9, 4].

However, students may not be completely sure of the submissions to their assignments. The uncertainty can lie in the syntax of the query, the edge cases the query handles, query complexity, or failure to understand the requirements only [10]. This affects their grades as the students are only evaluated based on single submissions. Consequently, they may receive lower grades, which undermines the intended learning outcomes of the assessment. This leaves no opportunity for the students to fill the gap between the requirements and the learning. Hence, there is a need for an automated assessment tool that could provide students with valuable immediate feedback [11].

This feedback could help the students understand the correctness of their submissions and would allow them the scope to correct their highlighted mistakes. It would

also allow the students to gain confidence in their submissions. This would help them to build good grades and provide them with a guided learning experience which is the ultimate goal of such lab exercises. Apart from helping students, this system would also allow teachers to gain an overview of how the student has performed on the given assessment. This would reduce the time and effort required to execute the queries manually and evaluate a submission. It would also be beneficial in identifying the learning gaps in students if any. This would allow instructors to design and lead the course in a direction that benefits students' learning curve.

In this research work, I have addressed two different sets of user groups:

- Individual students enrolled in database-related courses.
- Instructors and Graders that want to allow automatic evaluation of the student submissions to help them learn better and avoid mistakes to achieve better grades.

Similar attempts to auto-evaluate assignments have been made in other courses like programming assignments in Java using CodEval [2] and Scala using the system discussed in [12]. According to the statistics achieved from [2, 12], these tools have been beneficial for both students and instructors. There are other evaluation tools like TestSQL [3] and SQL Tester [13]. These tools also provide an evaluation system for SQL but have their own limitations discussed in details under Section 2.1.

The goal of this research paper is to design and develop a system that auto-evaluates the student's SQL-based assignments. The system would execute each submission in isolation from others using Docker containers. The system would be deployed in integration with CodEval [2] to provide feedback using a test cases-based model. The test cases would be designed to provide qualitative feedback for both students and instructors. Section 3 discusses the design of the EvalSQL in depth.

CHAPTER 2

History and Background

SQL is used in the design and retrieval of meaningful data from databases. The language consists of a set of protocols to follow. Following these, a user can fetch the required information from the database. This requires the user to learn the correct syntax to write the queries. There can be more than one way to write a particular query syntax. All such queries would produce the same result. Evaluating these multiple syntax queries makes the manual process more difficult [14].

S. Nayak, R. Agarwal, and S. K. Khatri provides a detailed survey of the positive impacts of automated assessment of SQL queries as seen in Figure 1 [1]. These positive aspects used for the survey are listed in Table 1. Due to such positive impacts on learning, there have been many initiatives to design a system for the automatic evaluation and grading of SQL queries [1].

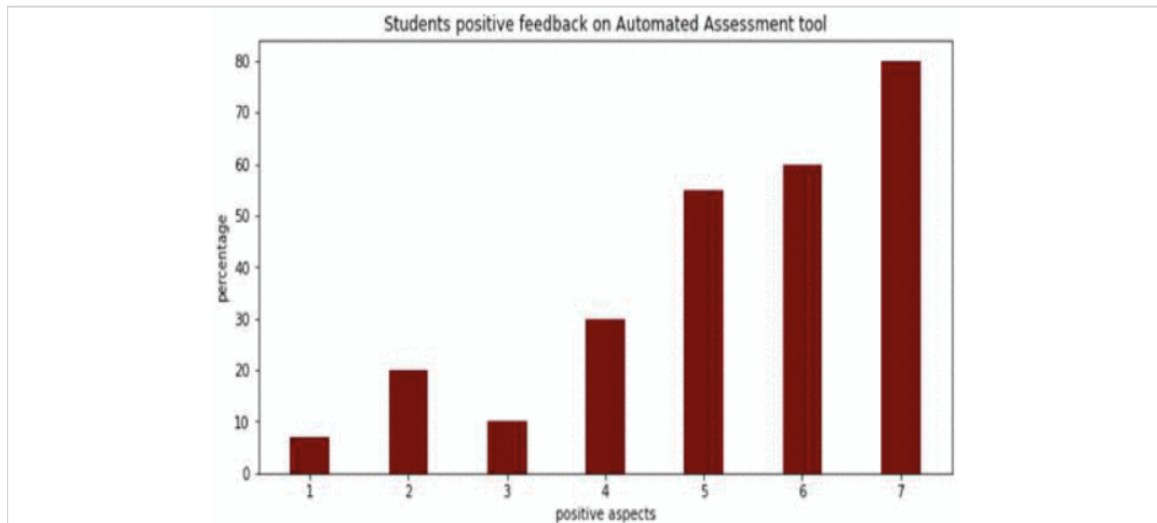


Figure 1: Positive aspect of the automated assessment tool. [1]

Positive Aspects	Description
1	Not Helpful
2	Time Management
3	Remote Submission
4	Creative Thinking
5	No Personal Instruction
6	Correct Grading
7	Immediate Feedback

Table 1: Describes the positive aspects considered in the above figure. [1]

2.1 SQL Automatic evaluation/ grading systems

In 2017, TestSQL was released as an open-source web-based platform to learn SQL interactively [3]. The system could import databases from a .sqlite format file. The system dynamically develops the questions on the database. The user can then run SQL queries for given questions on the database. These inputs were tested for the correctness of the query and the result data sets matched as shown in Figure 2. The system aligns more with interactive learning and could not be used for student assessment.

In 2018, A. Kleerekoper and A. Schofield [13] designed SQL Tester which is based on evaluating SQL statements. It was designed to evaluate simpler problems with no join clauses. It uses a syntax-based comparison to automatically assess student submissions. The student submission is compared to an instructor-defined answer and is graded only if it matches perfectly.

In 2020, authors F. A. K. Panni and A. S. M. L. Hoque [15] proposed a similarity-based grading system that checks the degree of similarity syntactically in the student submission to the reference answer curated by the instructor. The overall score is the sum of the positive score for the similarities found and the negative score differences. The combined score calculated, allows them to partially grade the

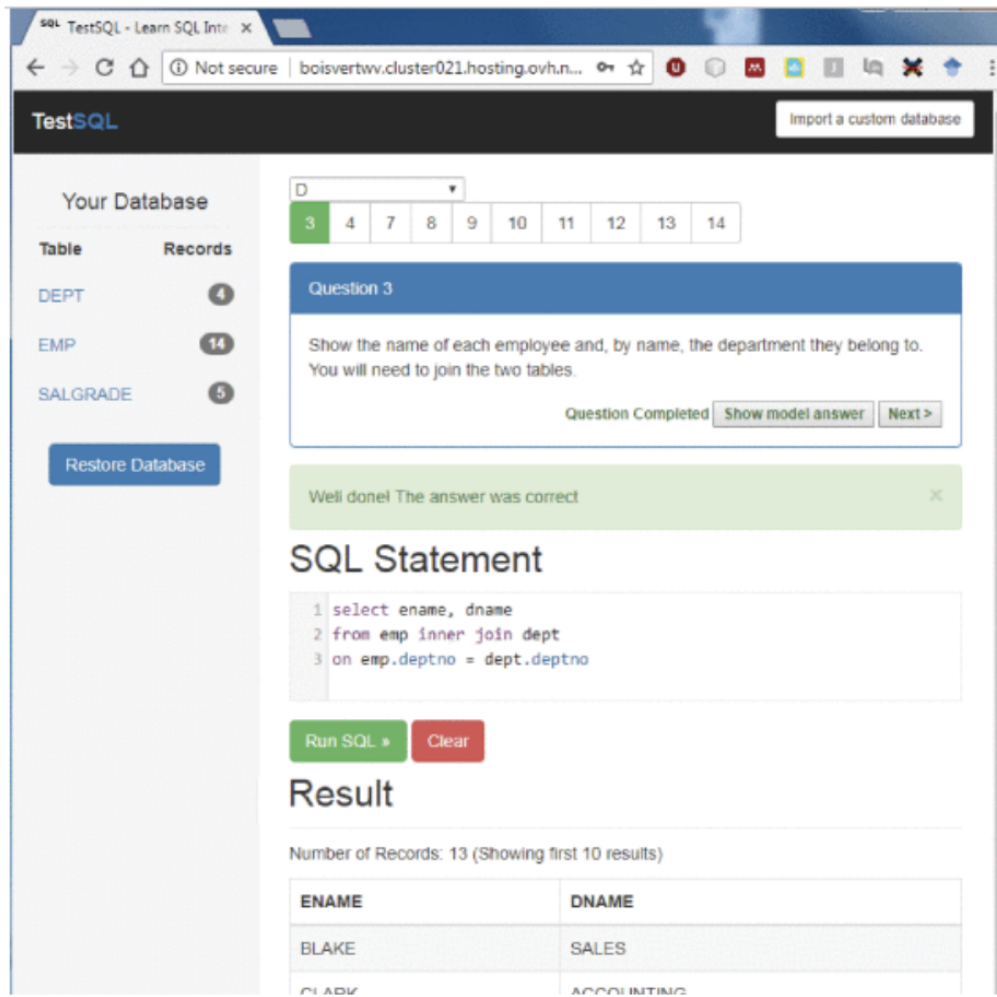


Figure 2: TestSQL User Interface. [3]

submissions which [13] did not consider.

S. Sadiq, et al. [16] created another platform that graded the submissions based on a model solution. The platform is interactive and students can submit solutions infinite times until the correct solution is reached.

Another model was developed by Chandra, et al. [17] that used a similar approach as [13, 16] but introduced partial marking schemes. The model answer from the instructor is compared with the submission if matched 100 percent full grades are

allotted. However, if there is a partial match the similarity to the instructor query is graded based on how close the data sets generated are.

Chandra, et al. [18] suggested another system for grading SQL queries. This model in contrast to [17] checks for the difference in the submitted query and model query. It grades the submission based on how many edits are required to match both queries' syntax.

The edits are defined as a minimum number of insertions, deletions, updates, and movements in the query. They defined it as the edit distance and used it to provide feedback and grades. The edit distance was calculated using the Shortest path algorithm and followed a heuristic approach. The system performed well with complex queries as well but required a lot of cost and time.

Another way for partial grading was introduced by [19] where the authors used string similarity metrics to grade SQL query partially. Firstly in their process, they removed any unwanted characters like ",", ";", "(,)", "=", "<", ">", "!", "+", "-", "**". They also eliminated aliases, new line characters with white spaces, and any leading/trailing spaces. Then further calculations like the smallest number of edits, the absolute difference in the model, and submission queries were calculated to provide feedback.

Authors in [4] propose an automated approach for evaluating the correctness of SQL queries. The system design as seen in Figure 3 consists of four main components: a query editor, an assessment rule engine, a query evaluator (lexical and syntax analysis), and a marks generator. The query editor inputs the query and converts the SQL query into a parse tree, which is then normalized to remove inconsistencies and standardize the query structure. The query evaluator then executes the normalized query and generates a result set. Finally, the marks generator compares the generated result set with the expected result set to determine the correctness of the query. The

proposed approach was evaluated on a benchmark data set and achieved high accuracy in identifying correct and incorrect queries.

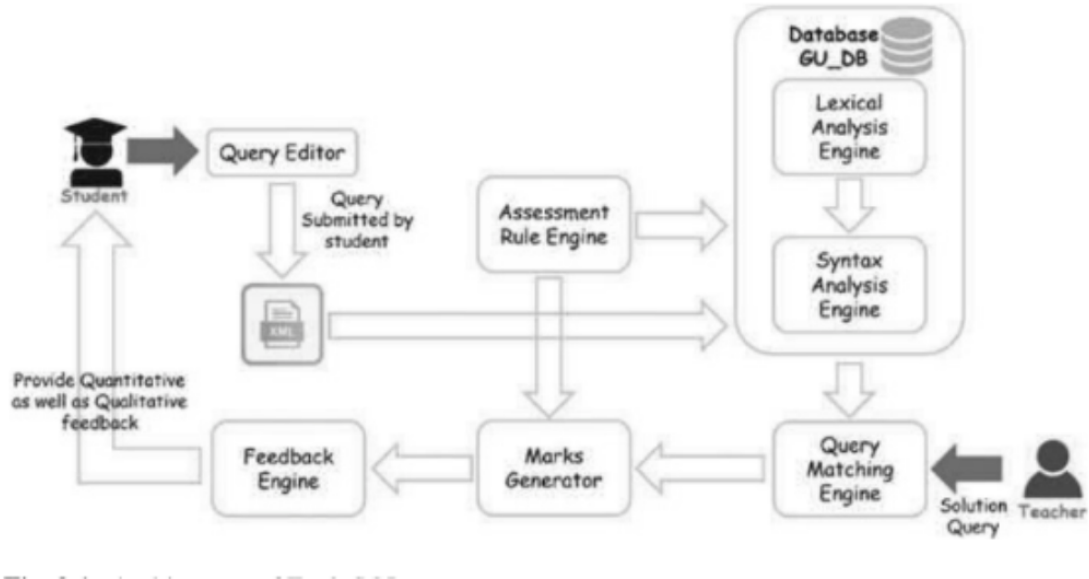


Figure 3: System Design for evaluating SQL [4]

The path to developing an automated grading system has been attempted by numerous research groups. However, there have been missing ends in designing a system that provides feedback to students not only based on the syntax but also on the usage of required keywords and provides a run time environment to evaluate if the solution executes correctly irrespective of the model answer. This will allow the instructor to evaluate the student submissions as close to manual evaluation as possible. Similar to approaches in [2, 12], the instructor can create test sets based on assignment grading rubrics for SQL. EvalSQL will use these test case-based executions to evaluate assignments. The tests defined will allow partial syntax matching, keyword matching, and run-time execution of the queries as collective feedback.

2.2 SQL Error Categorization

SQL queries include CREATE, INSERT, SELECT, UPDATE, and DELETE operations. They form the CRUD operations of SQL language. SQL also allows numerous other operations to write syntactically correct queries, including but not limited to GROUP BY, HAVING, and various types of joins. In total, there are seven distinct types of SQL queries that can be used to extract, manipulate, and analyze data from a database [14, 20].

When writing a SQL query two possible errors are likely to occur: syntax and logical errors [21]. Syntax errors are defined as system errors where the query does not execute as the structure is wrong for a DBMS to execute. Logical or semantic errors are errors in the logic of the query, the query executes without error but the data returned is not the expected result [22, 23]. Syntax errors are usually identifiable as they are highlighted by DBMS as well, but logical errors are challenging for students to fix as they occur due to incorrect/ limited understanding of the query requirements [24, 25].

Error ID and description	Freq.
missing expression	0.29
extraneous or omitted grouping column	0.18
missing join	0.10
missing column from ORDER BY clause	0.10
incorrect comparison. op. or incorrect value compared	0.08
omitting a join	0.06
<i>number of queries analyzed</i>	<i>3,739</i>

Figure 4: Most common error frequencies for all queries.

T. Taipalus [25] performs an analysis of 3800 incorrect queries to understand the further categorization of semantic errors. The result set as shown in Figure 4 and Figure 5 lists that missing expressions and extraneous or omitted column top the error charts in both single-table and multi-table queries. This identifies the need

Error ID and description	Freq.	Cause
missing expression	0.31	memory overload
extraneous or omitted grouping column	0.22	ignorance
missing join	0.12	absence of cue
incorrect comparison op. or value compared	0.08	ignorance
omitting a join	0.07	ignorance
missing column from SELECT	0.06	ignorance
<i>number of queries analyzed</i>	<i>2,869</i>	

Figure 5: Most common error frequencies for multi-table queries.

for the system to evaluate both logical and syntax errors from a query evaluation when considering the development of an automatic evaluation system. Syntax errors can be captured if the automatic evaluator is not able to execute the query and the system returns an error, and logical errors can be identified by extracting partial texts from queries and matching the same with assignment requirements from rubrics/instructors model query. A comparison between the existing and proposed system is detailed in Table 2

System	SQL Learn- ing	SQL query assess- ment	SQL query feed- back	Partial Evalua- tion
Esq1	Y	N	N	N
SQLator [17]	Y	Y	N	N
EvalSQL(Proposed System)	Y	Y	Y	Y

Table 2: Comparison for different tools.

CHAPTER 3

System Design and Methodology

EvalSQL is designed to auto-evaluate the Canvas-hosted SQL query-based assignments to provide timely feedback to the students. EvalSQL will be integrated with CodEval [2] to access, and download the student submissions from Canvas [26], and upload feedback back to the Canvas student account. CodEval [2] currently allows the test execution for programs written in C and Java. EvalSQL will extend this functionality and allow the evaluation of the SQL queries as well. EvalSQL would be executing student submissions on a dockerized platform. W. Z. Zhang and D. H. Holland [27] discuss one such approach and design for executing SQL queries on a container and providing a Database as a Service (DBaaS) [28].

3.1 Objective of EvalSQL

EvalSQL is designed to achieve the following objectives for two user types: the instructor and the student.

- Reduce manual efforts in evaluating student submissions for SQL-based database assignments. Currently, the system is proposed to evaluate SQL queries on the MySQL database.
- Assist students with immediate feedback to help keep the motivation to learn and excel.
- Allow students to understand the correct objective of the assignment questions and prevent them from making unwanted mistakes to help them grade better.
- Provide rubrics-based evaluation on student submissions.
- Give a complete learning experience to the students integrated with the Learning Management System LMS like Canvas.
- Assess SQL CRUD operations like INSERT, DELETE, UPDATE, and SELECT, etc.

- Allow execution of SQL queries in isolation of other students' submissions using Docker containers.

3.2 System Inputs

The system inputs will be the student submission file and a requirement file consisting of the tests to be executed on the submission. The requirement file will be created by the instructor consisting of the required tests as per assignment evaluation rubrics. The student is expected to submit a .sql file with a query or set of queries as required by an assignment question. For instance, for a query, as follows the contents of a student submission should be

Create a new table Employee with id (int) and name(varchar)



```
test.sql -- ~/Desktop
test.sql x
1 CREATE TABLE Employee(
2   id integer,
3   name varchar(30)
4 );
```

A test case file will be created for the above example query to provide qualitative and quantitative test feedback to students. The quantitative feedback would test the query from assignment rubrics defined to check the presence of expected keywords like CREATE, the field names: their types, etc. The qualitative feedback would be a test execution of the query in MySQL setup in a sandbox environment [29]. This would highlight any syntax errors, runtime errors, and discrepancies in the expected and actual results.

3.3 MySQL-based query execution on Docker containers

For the execution of the tests on assignments, the system requires a SQL environment setup for running and evaluating the queries. Most current solutions available for automatic evaluation for SQL queries only handle SELECT statements (as select statements don't execute any changes to the database schema) due to security issues like SQL injection. Authors in [29] discuss these issues with the existing tools and hence discuss their limitations in evaluating DDL(CREATE) and DML (INSERT, UPDATE) statements. These limitations are due to the risk of data injection, security issues, and data leaks. [29] also proposed a solution to execute the queries in a MySQL sandbox environment. Such environments support independent execution of queries with no risks of data corruption and relative. Hence to avoid such security issues and allow execution of DDL and DML statements, EvalSQL would execute in a MySQL sandbox environment. The following list discuss the steps EvalSQL would perform in such environment:

- Prerequisite setup: Download student submission zip files from a learning management system like Canvas.
- System setup: The system would run a docker environment providing an isolated environment for test set execution. The setup would involve configuring the MySQL server and client in the container. A separate database instance will be created for each submission to avoid any data leaks.
- Test Execution: Once the setup is completed, the test environment is ready for the execution of the test files. The test cases would be executed on the submissions using the command line interface to the container.
- Results: After the execution is completed, the test results are displayed on the canvas student submission in the comment section.
- Cleanup: The cleanup function would kill all the running processes and the

docker container as well. This would prevent any data leaks into other student evaluations.

For every student submission, the above steps would be repeated. This would support independent execution.

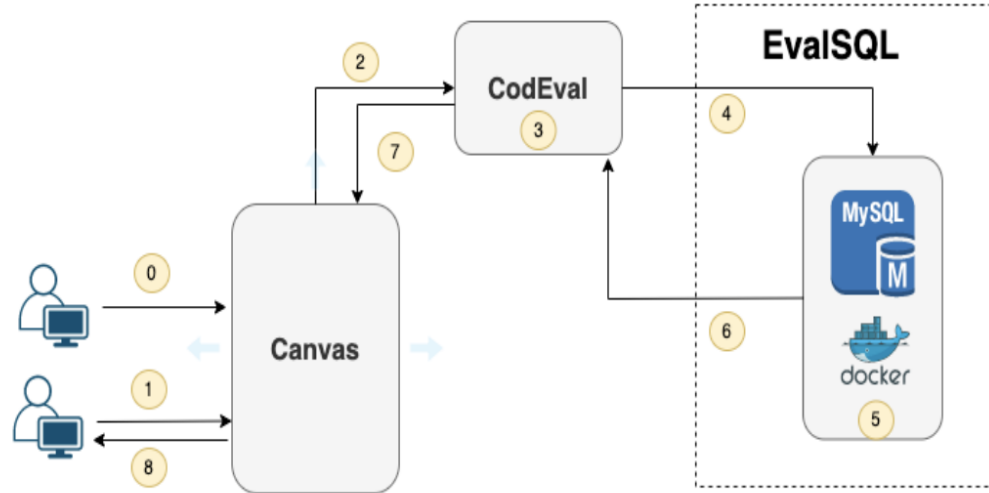
3.4 High Level Design

The system is built to run for submissions of LMS like CodeEval [2]. The instructor has to configure the test files for an assignment to be assessed by the system. The test files are stored on the LMS files, for the system to be fetched during test executions. EvalSQL integrated with CodeEval would download submissions and run them through the system for evaluation. After the complete test execution results are returned to the student as feedback that includes passed and failed tests. The student can use the feedback to improve their submissions if required until the solution completely passes all scenarios and is both syntactic and logically correct in all required ways.

The system architecture and high-level flow diagram are described in detail in Figure 6. This figure explains in detail the components involved in the automatic assessment. It also explains the steps executed as a complete solution to run and evaluate the assignments. When the student submits the assignment on Canvas the system starts its processing and provides constructive feedback to the student in comments as result.

3.5 Processing SQL queries

Processing SQL submissions from students is the core functionality of EvalSQL. The system runs predefined tests on the queries to analyze them and look for any deviation from the required query by the Instructor. SQL statements can be broadly classified into DDL, DML and DQL statements. The tests are based on the type of



0. Uploads assignment questions and CodeEval test file.
1. Assignment Submission
2. Download new submissions.zip
3. Run the codeeval.py execution process until a SQL block starts
4. Transfer control to the EvalSQL system, and start setup the MySQL server and client in a Docker Container
5. Execute tests on the submission file inside the container.
6. After complete execution EvalSQL cleanup and kill all related processes and return stored test results to CodeEval.
7. Post feedback in comments on Canvas.
8. Feedback available to a student.

Figure 6: System design with Canvas, CodeEval, and EvalSQL.

the SQL statement to be assessed.

3.5.1 Processing DDL statements

The DDL statements consist of CREATE, DROP, ALTER, and TRUNCATE. Such commands cause changes in the database structure and create/alter the relation schema and relation itself. Such statements do not return result data sets. To test the correctness of such statements EvalSQL would execute the query on MySQL and would validate the changes on the database to pass the test. For instance, consider a schema defined using Figure 7 and assignment questions can be:

Query 1 - Create the tables with the proper primary key and foreign key.

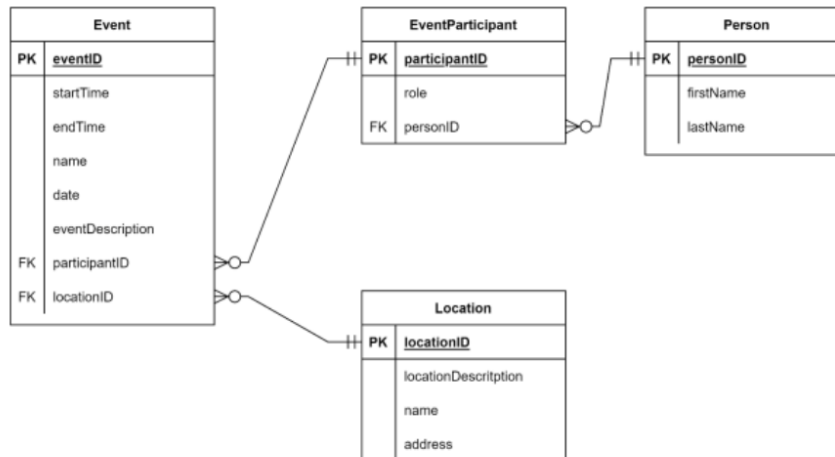


Figure 7: Sample Database Entity relationship diagram for assignment.

The test would include

- Running SQL statements to validate that the tables are created.
- Each table consists of the required number, type, and name of the columns.
- The integrity constraints are created as defined by the schema.
- The syntax errors to check for run-time errors.
- The tests are independent in nature and would return results as passed/ failed for each defined test. Together these tests would define the correctness of answers submitted by students.

3.5.2 Processing DML statements

The DML statements include INSERT, UPDATE, and DELETE. They update the state of a relation defined in a database. To evaluate such statements the relations data have to be fetched and analyzed based on the query. For instance:

Populate the database with at least 3 entries for each category. (Use SQL to

insert records). You will have to insert PK first before you can add a foreign key.

The test would include

- Running SQL statements to validate the row count of tables.
- Check syntax errors to include cases if there are any integrity constraint violations, as such queries will not execute and DBMS will throw run-time errors.

3.5.3 Processing DQL statements

The DQL statements include only SELECT. These statements can become very complex with the inclusion of multiple tables or nested queries. Another concern while writing these queries is the lack of requirement understanding and ambiguous statements. For instance, the following question is based on the Figure 6 schema.

All events that start at the same date (make sure you add two events with the same date in steps 3 and 4).

To validate the query, the tests would include-

- Syntax error detection for run-time errors. In such cases, the query fails to execute and no results are returned.
- When the query is syntactically correct, it returns a result dataset. The result set is stored in a file as actual output. The model will then execute the instructor query and store the result in another file as the expected output. The two files are then compared and any difference between the student submission and the instructor query implies the two result sets are different and the query does not perform as expected.
- The test can also include the presence of keywords to provide constructive feedback if some syntax error occurs. The feedback would highlight missing expressions and keywords that are expected to be.

The processing of DDL, DML, and DQL statements would allow EvalSQL to

generate test results as feedback of an initial automatic evaluation. This feedback would be made available to both instructor and student to help students submit better assignments and assist instructors in getting a first-eye review of student submission performance.

CHAPTER 4

Survey

4.1 Pre-execution survey

Before starting the tool-based evaluation We introduced the students to the project idea and carried out a survey to learn about how helpful EvalSQL would be on different parameters. Another objective was to check with students about additional features that would help them in validation for their submissions. Figure 8 shows the results of the survey.

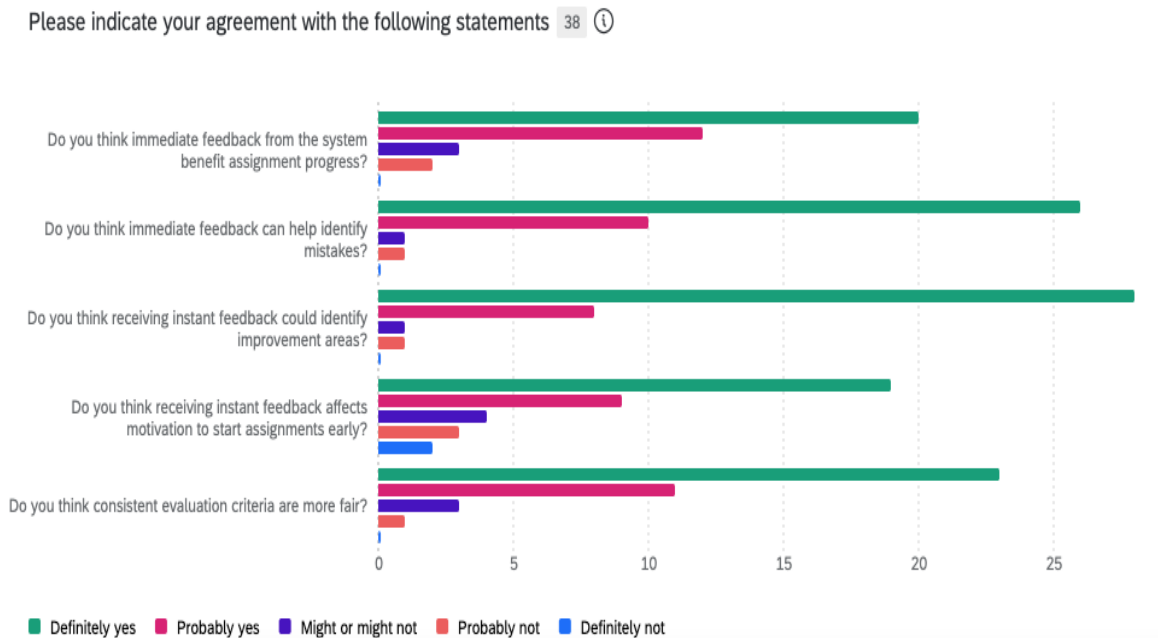


Figure 8: Survey results of the automated assessment tool.

The survey was conducted for two sections of a course with 100 students combined, and we received 47 submissions. Table 3 details the counts of votes for different metrics.

Based on the feedback received from the votes, it became apparent that students had a positive perception of the evaluation tool’s potential benefits, particularly in addressing two crucial project aspects: ”Helping to identify mistakes” and ”Identifying

Metric	Definitely yes	Probably yes	Might or might not	Probably not	Definitely not
Benefit assignment progress	20	12	3	2	0
Help identify mistakes	26	10	1	1	0
Identify improvement areas	28	8	1	1	0
Start assignment early	19	9	4	3	2
Consistent evaluation criteria	23	11	3	1	0

Table 3: Votes count in each category.

areas for improvement”. However, there were mixed responses in categories such as ”Facilitating assignment progress”, ”Encouraging early assignment initiation”, and ”Ensuring consistent evaluation criteria”. Upon analyzing the feedback derived from student votes, it became evident that the evaluation tool held significant promise in addressing key project requirements.

Subsequently, we initiated the execution of the auto-evaluation tool for three assignments in two sections

4.2 Post-execution survey

At the end of the semester, I conducted an additional survey to learn the tool’s performance and to gather insights on potential enhancements or suggestions from students aimed at improving their overall experience.

Figure 9, 10 and 11 show the survey results. From the figures it can be deduced that EvalSQL surely helped students in their assignments. It helped students to learn more from the assignment and also assisted them to find errors in their submissions. This allowed them to debug for errors and update their submissions to secure better grades. Following lists some of the quoted comments from survey:

- ”It helped us know what mistakes we made so we can actually fix them before fully submitting.”

- "Instant feedback for a better grade."
- "I was able to improve my code before it was actually graded and that way I was able to get a higher final score for those assignments."

Additionally, I asked students about some suggestions or improvements that would be beneficial for EvalSQL. There were some really good suggestions that will definitely allow EvalSQL to be more reliable and user friendly. Few of the answers are listed below and also included in Future work section.

- "It would be nice if the code was checked faster? If I'm not mistaken it currently only checks in 10 minute intervals, which would mean that we can only check if our code was correct 6 times every hour."
- "I noticed that if it fails a certain test case it stops. I think it would be more beneficial to be able to see the rest of subsequent test cases as well."
- "Less wait time and easier to read error messages"

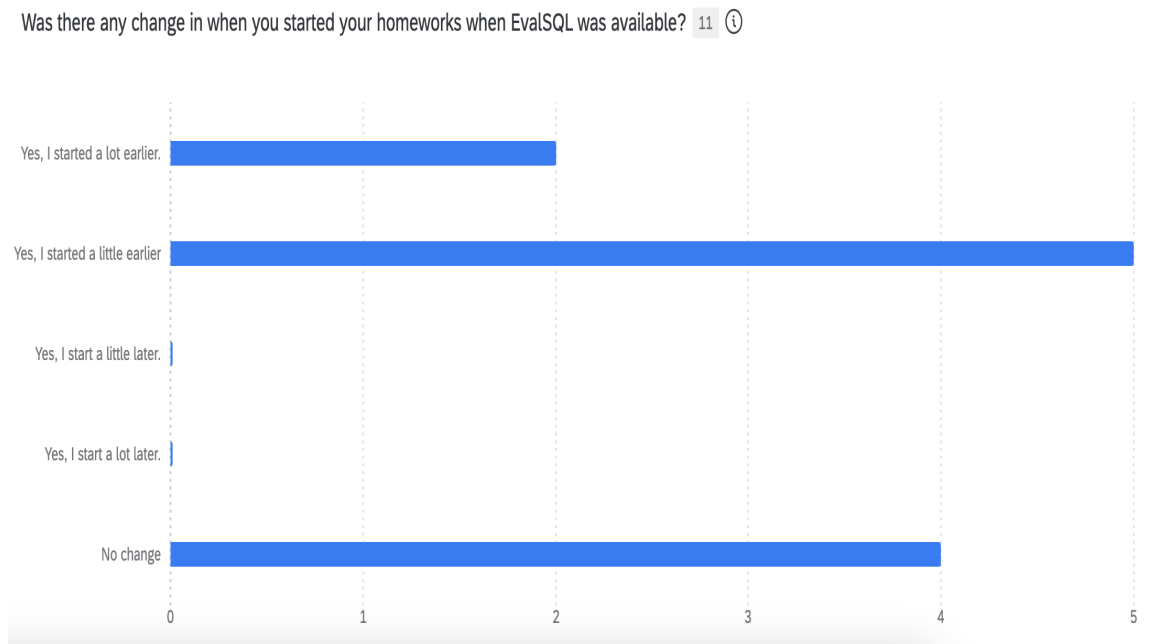


Figure 9: Survey question 1 of the automated assessment tool.

Was there any problem or code improvement that EvalSQL helped you identify? 11 ⓘ

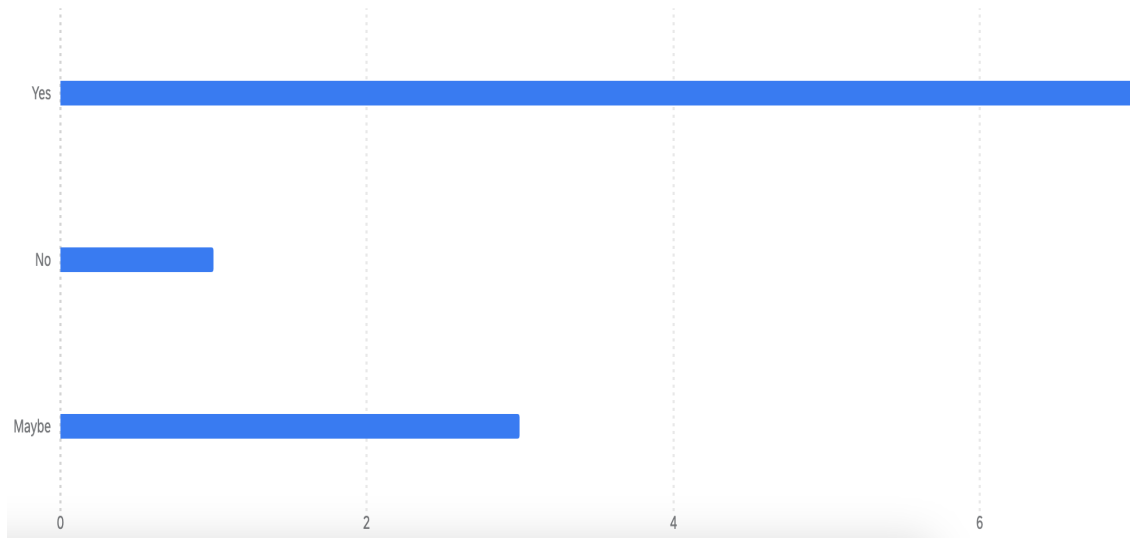


Figure 10: Survey question 2 of the automated assessment tool.

Did EvalSQL help you learn more about assignment? 11 ⓘ

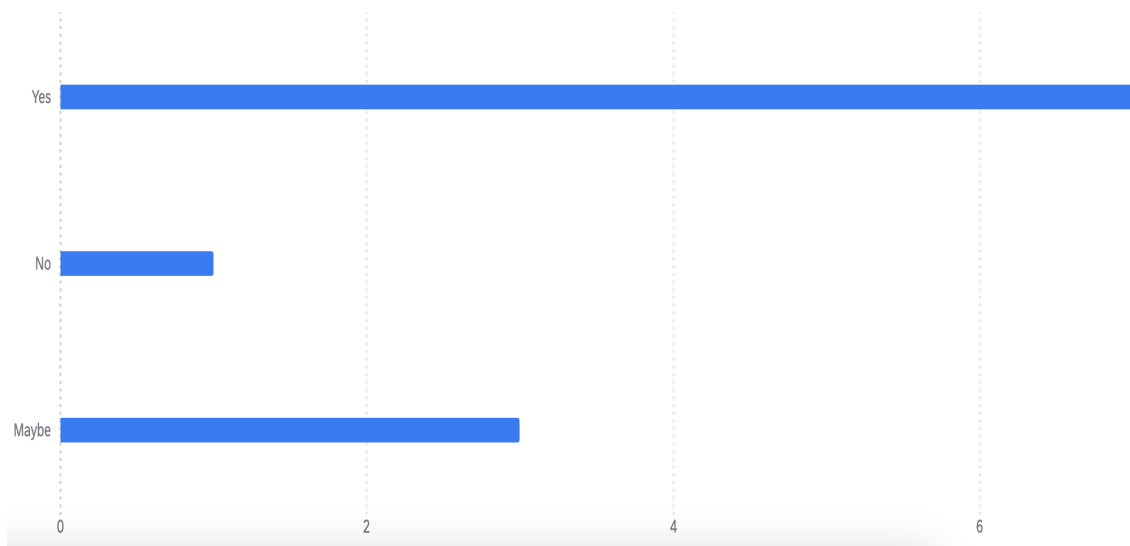


Figure 11: Survey question 3 of the automated assessment tool.

CHAPTER 5

Implementation

5.1 CodEval

EvalSQL is the added functionality for CodEval [2]: an auto-evaluation tool that allows instructors to evaluate student submissions on a predefined test set. The student benefit from the results as they can improve their assignments based on the evaluation results. The CodEval [2] has two major components that collectively execute the tests on student evaluation and provide them with feedback. The two components are specification files and the system built in Python.

5.1.1 The specification file

The specification file is the test file for the CodEval. It is a text file with .codeval extension. This file allows the instructor to write tests that need to be executed for the assignment. The file is easy to build and is based upon an easy underlying design. Figure 12 shows a sample specification file. The syntax of the file is easy to comprehend- it is the tag name followed by a space and then the input data associated with the tag. The input data can be a command, a file, etc based on the tag. For instance the tag "T" in figure 12 represents a test case where the input to T is a Java run command. CodEval has a directory of tags to do some predefined functions as explained in Table 4. For each assignment, a specification file is uploaded on the Canvas.

5.1.2 The Docker and the system

CodEval is built in Python language. It uses a containerized environment to run and execute submissions.

5.1.2.1 The Docker container

The Docker is a software that provides the platform to build, run and deploy application systems. The applications are executed in a containerized environment.

Tag	Meaning	Function
RUN	Run Script	Specifies the script to use to evaluate the specification file. Defaults to evaluate.sh.
Z	Download zip	Will be followed by zip files to download from Canvas to use when running the test cases.
CF	Check Function	Will be followed by a function name and a list of files to check to ensure that the function is used by one of those files.
CMD/TCMD	Run command	Will be followed by a command to run. The TCMD will cause the evaluation to fail if the command exits with an error.
CMP	Compare	Will be followed by two files to compare.
T/HT	Test Case	Will be followed by the command to run to test the submission.
I/IF	Supply Input	Specifies the input for a test case. The IF version will read the input from a file.
O/OF	Check output	Specifies the expected output for a test case. The OF version will read from a file.
TO	Timeout	Specifies the time limit in seconds for a test case to run. Defaults to 20 seconds.
E	Check error	Specify the expected error output for a test case.
X	Exit code	Specifies the expected exit code for a test case. Defaults to zero.
HINT	Hint	Provides hints to the hidden test cases.

Table 4: Specification tags in CodEval. [2]

The containers are light weight and isolated structures that allow the application to run in isolation with its own compute and memory. These containers are built using docker image files. The docker image files are created to build the compute and environment to run applications. CodEval uses these docker containers to run the

```

RUN evaluate.sh
C javac edu/sjsu/CS001/Hello.java

T java edu.sjsu.CS001.Hello ben
X 0
O Hello ben

T java edu.sjsu.CS001.Hello
X 1
O USAGE: edu.sjsu.CS001.Hello name

HT java edu.sjsu.CS001.Hello "long name here!"
X 0
O Hello long name here!

```

Figure 12: Specification file for CodEval. [2]

system and evaluate assignments.

5.1.2.2 The system

Another design aspect of the CodEval is the system that is responsible for the execution of the tags in specification files, creating a runtime environment, and allowing integration with Canvas. The CodEval system is built in Python language and also uses shell scripts to run submissions in a containerized environment. The steps in execution are as follows.

- The entry point to the system is the codeval.py file. This file is responsible for connecting to Canvas and fetching all the assignments for a given course as input.
- The specification file for the assignment, the submissions that are not evaluated yet, and supporting files are copied in a directory on the server.
- This volume is then mounted to the Docker container and the container is built

using the Docker image specified in the ini file.

- The Docker container starts and the shell script begin its evaluation in the container
- After the test execution completes the results are then displayed to the Canvas as a comment on student submission.
- The docker container stops and exits.
- All the above steps are repeated for each submission of an assignment that is not evaluated yet.
- After all evaluation is completed the execution is marked completed.
- The above process is scheduled for every 10 minutes via Cron jobs.

The EvalSQL is built using the CodEval system with added functionality to support SQL-based executions. To support SQL execution some updates were required that are discussed in Section 5.2 and Section 5.3.

5.2 The Docker image updates

Docker image files are standalone files that are used to create Docker containers. The docker container is a running instance of the application including the application and its dependencies. The Docker file used to create the image for the container-based executions in CodEval did not provide MySQL environment support. To set up the MySQL environment in the container a new docker image file was created. Figure 13 displays the docker image file. The Docker image is used to build a container that has MySQL setup to support EvalSQL to run and test submissions.

The docker file uses an Ubuntu base image 20.04. It then installs MySQL Server and some additional dependencies. The software-properties-common and add-apt-repository universe are used to enable the "universe" repository, which may contain additional packages. It then creates a MySQL client that is used to connect with the

MySQL server from within the container through the exposed port. Then it starts the MySQL client service and creates the database user "dummy" and a database "dummy". Further, it then grants all permissions to the user to perform SQL operations over all tables and views in the database. This allows seamless execution of the SQL runs as the user has all permissions within the default database "dummy". After establishing the user and database, the subsequent section designates them as the default options. In the standard configuration, upon executing the submission, it employs the "dummy" database and the "dummy" user for their operations.

The Docker image is configurable and can be updated to allow support to other platforms like SQL Server, PostgreSQL, and more.

5.3 Specification file updates

The specification file is a .codeval extension file that lists all the tests to be executed on student submissions. EvalSQL, an extension to CodEval required additional tags that would allow the execution of SQL-based DDL, DML, and DQL executions. The new tags were created and it allow the successful execution and evaluation of student submissions.

The SQL-based tags and their meaning are briefly described below in Table 5.

The **--SQL--** tag marks the beginning of the SQL tests in a CodEval specification file. The tag allows the CodEval system to understand that the specification file consists of SQL-based tests. It instructs CodEval to create a Docker container capable of running and testing SQL statements.

The **TSQL** tag defines a SQL test. It allows CodEval to interpret the test to be executed as not a regular test. This allows the execution of the student submissions on a SQL Docker container. The SQL Docker container container is capable of executing MySQL language statements. The TSQL tag can execute single MySQL statements

```

FROM ubuntu:20.04

# Set environment variables to non-interactive mode for MySQL installation
ENV DEBIAN_FRONTEND=noninteractive

# Install MySQL Server
RUN apt-get update && \
    apt-get install -y mysql-server && \
    apt-get install -y software-properties-common && \
    add-apt-repository universe && \
    apt-get update && \
    apt-get install -y gcc valgrind|

# Expose the MySQL port (default is 3306)
EXPOSE 3306

# Install MySQL client to interact with MySQL server
RUN apt-get install -y mysql-client

# Start MySQL when the container starts
#CMD ["mysqld"]
CMD ["sh", "-c", "service mysql start && sleep 5 && mysqld"]

# Create the 'dummy' MySQL user during container initialization

RUN service mysql start && \
    mysql -e "CREATE DATABASE IF NOT EXISTS dummy;" && \
    mysql -e "CREATE USER 'dummy'@'%' IDENTIFIED BY 'dummy';" && \
    mysql -e "GRANT ALL PRIVILEGES ON dummy.* TO 'dummy'@'%';" && \
    mysql -e "FLUSH PRIVILEGES;"

# Set the environment variables for MySQL user, password, and default database
ENV MYSQL_USER=dummy
ENV MYSQL_PASSWORD=dummy
ENV MYSQL_DATABASE=dummy

```

Figure 13: Dockerfile for EvalSQL container.

and complex SQL files as well. The file and the statements are executed in a MySQL environment where they are validated syntactically. Upon successful execution the

Tag	Meaning	Function
--SQL--	SQL Tests Begin	Marks the beginning of SQL tests. Is used to determine if the spec file has SQL tests.
INSERT	Insert rows in DB	Insert rows in the SQL database using files/individual insert queries.
CONDITIONPRESENT	Check conditions	Validate submission files for a required condition to be present in submissions.
SCHEMACHECK	Check constraints	Validate submission files for database-related checks like constraints.
TSQL	SQL Test	Marks The SQL test takes input as a file or individual query and runs it on submission files.
TSELECT	SQL DQL Test	Marks the SQL test, take input a student file, expected query file and run both and compare the outputs.

Table 5: Specification tags in EvalSQL.

test case passes however, if the syntax is incorrect the test fails with an error. The error is returned by the system and displayed in comments on the Canvas submission. The error displays the differences in expected and actual output or error.

The TSQL tag can be used in combination with the **O**/**OF**/**E** tags to define expected outputs or errors. For example, the O tag is used to test the expected output after execution of the TSQL test. Figure 14 represents a specification file that displays the syntax of using the TSQL tag. The first usage is only to check syntactically the file 1.sql with no expected outputs. The second usage is TSQL with an O tag where 2.sql is executed and the output is compared with the expected output defined in the O tag. The last is also a combination of TSQL and O tag with TSQL executing a single statement.

The **TSELECT** tag defines a SQL SELECT-based query test. It is used to test

```

--SQL--
TSQL 1.sql
INSERT insert.sql
TSQL 2.sql
0 count(*)
0 3
TSQL DELETE from Person;
0 count(*)
0 0

```

Figure 14: TSQL tag implementations.

DQL statements. The tag takes the student submission file and the expected file as inputs. The student submission file is the file submitted by the student whereas the expected file here contains the sample solution for the assignment. This sample solutions are drafted by the instructor or the grader. EvalSQL executes both files in the container on the same database and then compares the output and error files to find differences. Any differences between the output files mark the test case as failed. If both files produce the same result set the test case is passed. Passed or failed the results for the test are displayed for the student on Canvas. Figure 15 represents an example specification file that shows the usage of the TSELECT tag.

```

--SQL--
TSQL Hw4/create.sql
TSQL Hw4/insert.sql
TSELECT Hw4/b.sql bexpected.sql
TSELECT Hw4/c.sql cexpected.sql
TSELECT Hw4/h.sql hexpected.sql
TSELECT Hw4/l.sql lexpected.sql

```

Figure 15: TSELECT tag implementations.

The **INSERT** tag is used to insert records in the database. In SQL rows can be inserted using file uploads and using INSERT statements with VALUES command.

The data can be inserted using files where the student or instructor can add multiple INSERT statements in an SQL file to perform bulk operations. It also allows the insertion of a single row. The instructor can use the tag to INSERT data and validate student submissions over it.

With **CONDITIONPRESENT** tag the instructor can check whether a required condition is present in a SQL submission. It can also be used to validate that the SQL submission contains a required value. This allows the instructor to test for additional checks in the submission. Figure

```
SCHEMACHECK PRI Person P_id
SCHEMACHECK PRI Location L_id
CONDITIONPRESENT count 2.sql
CONDITIONPRESENT Person 2.sql
```

Figure 16: SCHEMACHECK and CONDITIONPRESENT tag implementations.

The **SCHEMACHECK** tag tests the submission's DDL and DML statements. The tables and views in a database are created and updated with a set of columns. The columns can have constraints (additional rules) that limit the data values a column can hold. These constraints can be added to column definitions using DDL and DML statements. Table 6 lists the SQL constraints and the keyword used with the tag to check if a column has that check in a table or not. 16 represents an example specification file that shows the usage of the **CONDITIONPRESENT** and **SCHEMACHECK** tags.

SQL Constraint	SCHEMACHECK tag Keyword
Primary Key	PRI
Unique	UNI
Not null	MUL
Foreign	forn
Default	DEFAULT

Table 6: List of SQL constraints and associated keywords in SCHEMACHECK tag.

CHAPTER 6

EvalSQL Execution

The EvalSQL execution requires to make some setup changes to the system. The changes required are described in the below sections.

6.1 Setup

To run the EvalSQL, firstly a configuration change is required. The configuration change is for the docker image that needs to be updated if the specification file contains SQL tests. The updated configuration file is displayed in Figure 17. CodeEval identifies a SQL test specification file with the presence of `--SQL--` tag. If the tag is present in the file, the system runs a different container image that is created using the docker file described in Section 5.2. Secondly, the instructor needs to generate a new Canvas token and add it to the configuration file. The instructor must also update the canvas URL for the university.

```
[SERVER]
url=<CANVAS URL HERE>
token=<CANVAS TOKEN HERE>
[RUN]
precommand=
command=docker run -i -v SUBMISSIONS:/submissions -v ~/.m2:/root/.m2 autograder-java bash -c "cd /submissions; EVALUATE"
; The following lines are needed for distributed assignments autograding:
dist_command=docker run --name NAME -dt -v SUBMISSIONS:/submissions -v ~/.m2:/root/.m2 PORTS autograder-java
host_ip=<your machine IP>
; The following lines are needed for sql assignments autograding:
sql_command=docker run -i -v SUBMISSIONS:/submissions -v ~/.m2:/root/.m2 autoeval-sql bash -c "service mysql start >/dev/null 2>&1 disown; cd /submissions; ./checksql.sh; EVALUATE"
```

Figure 17: Configuration file for EvalSQL.

6.2 Specification file creation

After the configuration changes are completed the next step is to prepare the specification file for the assignment. The specification file can be created using the CodeEval and EvalSQL tags listing in section Section 5.1.1 and Section 5.3. An example specification file is displayed in Figure 18. The specification file starts with the execution of file `evaluate.sh`. This file is a shell script that contains an implementation of all the tags for CodeEval and EvalSQL. The next tag is `--SQL--` that specifies the assignment is SQL-based and launches a SQL docker container. The Z tag is used to

unzip and add files in insert.zip to the container storage mount. The TSQL, INSERT tag is used to test the execution of student submission files. The O tag is used to define the expected output that the TSQL command generates. The SCHEMACHECK validates the primary key for table Person is P_ID and CONDITIONPRESENT tags then checks for count present in the input file.

```
RUN evaluate.sh
--SQL--
Z insert.zip
TSQL 1.sql
INSERT insert.sql
TSQL 2.sql
O count(*)
O 3
TSQL select count(*) from Person;
O count(*)
O 3
SCHEMACHECK PRI Person P_id
CONDITIONPRESENT count 2.sql
TSELECT Hw/student.sql expected.sql
```

Figure 18: Sample Specification file for EvalSQL.

Once the file is created, the specification file should be uploaded in the Canvas

Files section within the CodeEval folder as seen in Figure 19. The file name should be the same name as the assignment to be graded.

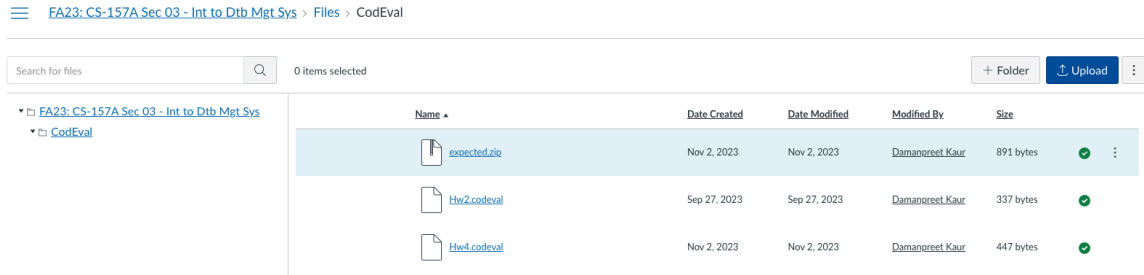


Figure 19: Specification file Uploaded for EvalSQL.

6.3 Running EvalSQL

To schedule the test executions for a course, the codeval.ini should have an updated canvas URL and token. The Canvas token created and used should belong to the instructor or the teaching assistant roles. These roles in Canvas have grading permissions and access to student submissions. EvalSQL uses the same as CodeEval to run.

```
python3 codeval.py grade-submissions COURSE_NAME OPTIONS
```

The entry point to EvalSQL is codeval.py (grade-submissions method), which is responsible for Canvas interactions through Canvas REST API and executing the Docker container to run specification file tests. The next required argument is the course name or a unique identifier to the course for which the assignment is to be graded. There are a few options that can be included in the run statement to support the execution as follows

- `--verbose/--no_verbose` It is used for detailed/limited logging for every step during execution.
- `dry run/no_dry_run`: It allows to run the entire process with/without updating the feedback on Canvas respectively.

- force/no_force: It will allow/ not allow grading even if the assignment is already graded.

Figure 20 displays the two different ways to schedule and run EvalSQL. The first way is through the crontab job scheduler where the job is scheduled to run in a timely interval (for example 10 10-minute intervals as seen in the screenshot). This is an automated execution where the instructor creates a crontab job and runs it until the assignment is due.

The second way is manual execution. Manual execution is a one-time run. The same command can be used to run it. When the command is entered it searches for all the submissions that are yet to be graded for the input course. The assignments are evaluated and the script stops once the execution is completed (no remaining evaluated assignment). The logs are printed on the console. This way is ideal for debugging issues or during upgrades to test the functionality.

```

REQUESTS_CA_BUNDLE=/home/daman/CodEval/autograder/certifi/cacert.pem
0,10,20,30,40,50 * * * * cd /home/daman/CodEval && python3 /home/daman/CodEval/codeeval.pyz
grade-submissions --verbose --no-dry-run 'CS-157A Sec 03' >> /home/daman/tmp/codeeval.out 2>
&1
0,10,20,30,40,50 * * * * cd /home/daman/CodEval && python3 /home/daman/CodEval/codeeval.pyz
grade-submissions --verbose --no-dry-run 'CS-157A Sec 01' >> /home/daman/tmp/codeeval.out 2>
&1
# 0,10,20,30,40,50 * * * * cd /home/daman/CodEval && python3 /home/daman/CodEval/codeeval.py
z grade-submissions --verbose --no-dry-run 157A >> /home/daman/tmp/codeeval.out 2>&1
daman@cs-reed-07:~$ python3 codeeval.py grade-submissions PracticeCourse_Reed_2]

```

Figure 20: Sample Scheduling Jobs for EvalSQL.

6.4 Evaluating submissions

When the job is scheduled for execution, the next step is the evaluation. The job is scheduled to run every ten minutes. After a student uploads their submission the EvalSQL and CodEval executes the student submission and return the results as feedback in the comments. The comment details the execution results for each test from the specification file. If a test fails the execution for that submission is stopped and the result is returned and displayed in Canvas. Figure 21 shows an example

run for a student. The specification file is built with a total of six tests. For this example, the third test failed, and the failure reason for its failure is also returned to the student. The failure reason includes the following.

- The command executed that caused the failure.
- The lines that are present in the actual output and not in the expected output are prefixed with '-'.
are prefixed with '-'.
- The lines that are present in the expected output and not in the actual output are prefixed with '+'.
are prefixed with '+'.

The detailed output helps the student identify the issue in their submission and re-submit the updated and corrected assignment. For this example, the command run should return the number of records in the Product table. The command is expected to return 30 however it returns 22 count. This indicates that the Product table is missing some values and the student can check the inserted data to correct their submission.



[AG]


MySQL is running.
MySQL configuration file (/root/.my.cnf) created successfully.
Testing submission files

Test Case 1 of 6: Passed

Test Case 2 of 6: Passed
Testing Inserted Data counts

Test Case 3 of 6: FAILED
Command ran: "select count(*) from Product;"
The lines prefixed with '-' are wrong output lines present in your output and not present in the expected output.
The lines prefixed with '+' are present in the expected output and should be present in your output.
Your output file: ./youroutput^
Expected output file: ./expectedoutput^
count(*)\$
-22\$
+30\$

Figure 21: Failed execution for EvalSQL.



[AG]
MySQL is running.
MySQL configuration file (/root/.my.cnf) created successfully.
Testing submission files

Test Case 1 of 6: Passed

Test Case 2 of 6: Passed
Testing Inserted Data counts

Test Case 3 of 6: Passed

Test Case 4 of 6: Passed

Test Case 5 of 6: Passed

Test Case 6 of 6: Passed
took 0 seconds

Figure 22: Passed execution for EvalSQL.

After updating and resubmitting, the evaluation is performed again. Figure 22 shows that the test cases passed and the student is able to view another comment from EvalSQL and CodeEval. The evaluation is now completed and student can rest assured about the correctness of their assignment.

CHAPTER 7

Difficulties and Challenges

During the design and development of EvalSQL, several challenges were encountered. These challenges encompassed a variety of issues, from having a positive user experience to designing a resilient and robust system. The following section discusses some of the encountered challenges.

7.1 Building MySQL server and client in one Docker container

CodEval[2] was designed for the execution of Java and C programming assignments. It did not allow SQL evaluations. EvalSQL is designed to extend CodEval functionality to allow SQL executions for the MySQL database. MySQL requires a client and server [30] to create databases and execute SQL queries. The docker image provided by MySQL is separate for the client and the server. To create one image for the server and client, a base UBUNTU image was used. On this base image, the MySQL server and client were installed through the Docker RUN command. The commands to start the client and server were also included in the Docker image. When the docker container starts, the container has an active instance for MySQL running.

7.2 Execute multiple scripts on docker container start

When a docker container is launched a default `entrypoint.sh` shell script is executed. This script creates the MySQL services for the execution as defined in the docker image. After the container is created the MySQL services should be started with the docker run command. However, the issue arose when the process for MySQL started in the main shell and it did not further allow `evaluate.sh` to run as the main process. As a result, the script was never executed and requests got timed out. To solve the issue and run MySQL on container start-up, the commands were added in the `-bash` option of docker run statements. The MySQL was started and the process was disowned to run in the background making it possible for `evaluate.sh` to run as

the main process and start with test executions.

7.3 MySQL user permissions to access LOAD statement files

The MySQL docker image that is used in EvalSQL did support all the SQL statements except one - the LOAD statement. The LOAD statement allows to upload of file contents to a table in the database. This file is also submitted as part of the assignment. Functionally CodEval downloads all submission files into a directory and then mounts this directory to the Docker container while creation. This allows the submission files to be accessible within the container as well. Similar to all files the data file for the LOAD statement was also added to the same directory. However, the issue emerges when the MySQL engine expects the file in a default path or requires the user permissions to be updated to access any external location. The issue was resolved by adding the path to the MySQL user's permissions.

7.4 Individual files for SQL statements

During the design of EvalSQL, the major challenge was how to accept the solution files so that it would be easy for students to locate errors, and update their submissions to pass all test cases. The possible ways were a single file for all parts or separate files for each part. Having one file is compact and easy to build, however, this would be difficult to evaluate and debug for errors. The final design decision was to have separate files for each part to allow the system to perform targeted tests and help students easily debug for errors.

CHAPTER 8

Limitations and Future Works

EvalSQL design is kept simple and extensible and can be used as a foundation to build more complex software. The following sections discuss in detail the limitations and potential avenues for future enhancements of EvalSQL to expand its functionalities.

8.1 Extending support to other database systems

EvalSQL supports the execution of SQL queries for the MySQL database. It allows execution and evaluation for DDL, DML, and DQL statements. This support can be extended to other database systems like PostgreSQL [31] and Oracle [32]. Both these databases are relational databases and similar to MySQL. These databases can use the basic functionality from EvalSQL and CodEval systems and extend them as well.

The updates would involve creating a docker image to build containers that support PostgreSQL and Oracle environments, creating its users to run the evaluation system, and adding new tags in specification files to differentiate between different databases. The possible tags could be `--MySQL--`, `--PostgreSQL--` and, `--OracleSQL--` instead of `--SQL--` tag. Each tag can have an associated docker run command in the configuration file.

8.2 Adding JDBC support

EvalSQL is currently used to run and evaluate the SQL queries for the MySQL database only. During the execution of the assignments, there were some assignments that required the students to build a Java-based application that connects with the database through JDBC drivers and interacts with databases. The JAVA program performs all statements including DDL, DML, and DQL through the support of drivers. The support for these assignments can also be added to the EvalSQL.

This would require creating an updated docker image that contains a JAVA

and MySQL environment to build and run these programs. With the update, an application can be tested end to end with Java using CodEval and a database using EvalSQL specification tags.

8.3 Adding scores to evaluation

Another proposed update from a faculty was to associate scores with the evaluation. CodEval and SQL are evaluation tools as they provide feedback to students. Grading still requires a manual effort, where the instructor or grader adds scores based on the evaluation results.

To update and reduce manual efforts of grading similar to the design of the rubric, each test can hold a percentage of the grade for the part it tests for. The software would then return points that are associated with each passed test in addition to the feedback text. The grade points can be returned for each test, a total score, or both.

8.4 Displaying detailed feedback outputs

The purpose of automatic evaluation is to provide immediate feedback to students to help them learn better and succeed in the learning outcomes for a course. One crucial part of such software is the feedback details that are returned to the student. The quality feedback should be easy to understand and provide correct and meaningful errors. The EvalSQL uses the feedback mechanism provided by CodEval currently.

To enhance it and make it more appropriate for SQL errors, we can potentially make a mapping of the errors and a text that can be returned to the student as feedback in addition to the SQL error codes that are returned. This would reduce the time spent by students to find and search for the error code in MySQL documentation. It will also be easy to read and debug the errors.

8.5 Breaking down the queries to identify logic build

EvalSQL provides an execution environment to run and evaluate SQL queries. The SQL queries can be broken down from complex queries to smaller and easier queries. These queries can be used to test the logic and provide partial feedback as well. It could help the system identify the part of the query that is not as expected output.

This would need designing an algorithm to break down queries like relational algebra binary tree-like structures. It would definitely require more research to identify the algorithm.

CHAPTER 9

Conclusion

Database design and implementation is a core subject in Computer Science, Data Science, and Data Analytics degree courses. Students enroll to learn and build knowledge about database design and execution. In learning databases, assignments are one core part of developing the skills. The assignments include designing databases and performing operations on them using databases like MySQL.

EvalSQL is built with the purpose of database-related courses to benefit from auto-evaluation systems like CodEval. EvalSQL extends the functionality in CodEval to allow auto evaluation of database design and implementation-based assignments. It benefits both students and instructors (including graders). Students are able to receive feedback on how their solutions performed against a pre-defined set of test cases designed by the instructor. These test cases closely resemble the grading rubric used by the instructor and the grader. It allows a smooth learning curve and helps build knowledge by providing immediate feedback to students. This allows students to build quality assignments and receive better grades. Instructors and graders also benefit from the results as it provides them with an overview of how the students performed. EvalSQL helps achieve one common goal of quality learning for students and instructors.

LIST OF REFERENCES

- [1] S. Nayak, R. Agarwal, and S. K. Khatri, “Review of automated assessment tools for grading student SQL queries,” in *2022 International Conference on Computer Communication and Informatics (ICCCI)*, 2022, pp. 1–4.
- [2] A. Agrawal, A. Jain, and B. Reed, “CODEVAL: IMPROVING STUDENT SUCCESS IN PROGRAMMING ASSIGNMENTS,” in *EDULEARN Proceedings*. IATED, jul 2022.
- [3] J. License, “testsql: Learn sql the interactive way,” 06 2017, pp. 376–376.
- [4] B. Shah and J. Pareek, *Automated Evaluation of SQL Queries: Eval_SQL*, 04 2022, pp. 89–104.
- [5] W. Hung, D. H. Jonassen, and R. Liu, “Problem-based learning,” in *Handbook of research on educational communications and technology*. Routledge, 2008, pp. 485–506.
- [6] M. Barg, A. Fekete, T. Greening, O. Hollands, J. Kay, J. Kingston, and K. Crawford, “Problem-based learning for foundation computer science courses,” *Computer Science Education*, vol. 10, 08 2000.
- [7] M. Sastry, “An effective approach for teaching database course,” *International Journal of Learning, Teaching and Educational Research*, vol. 12, pp. 53–63, 07 2015.
- [8] T. Taipalus and V. Seppänen, “SQL education: A systematic mapping study and future research agenda,” *ACM Transactions on Computing Education*, vol. 20, 05 2020.
- [9] C. Kleiner, C. Tebbe, and F. Heine, “Automated grading and tutoring of sql statements to improve student learning,” in *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling ’13. New York, NY, USA: Association for Computing Machinery, 2013, p. 161–168. [Online]. Available: <https://doi.org/10.1145/2526968.2526986>
- [10] A. Borthick, P. Bowen, D. Jones, and M. Tse, “The effects of information request ambiguity and construct incongruence on query development,” *Decision Support Systems*, vol. 32, 11 2001.
- [11] P. Brusilovsky, S. Sosnovsky, M. Yudelso, D. Lee, V. Zadorozhny, and X. Zhou, “Learning sql programming with interactive tools: From integration to personalization,” 01 2010.

- [12] A. Patil, “Automatic grading of programming assignments.” SJSU, 2010. [Online]. Available: https://scholarworks.sjsu.edu/etd_projects/51
- [13] A. Kleerekoper and A. Schofield, “Sql tester: an online sql assessment tool and its impact,” 07 2018, pp. 87--92.
- [14] G. Russell and A. Cumming, “Automatic checking of sql: Computerised grading,” 07 2005.
- [15] F. A. K. Panni and A. S. M. L. Hoque, “A model for automatic partial evaluation of sql queries,” in *2020 2nd International Conference on Advanced Information and Communication Technology (ICAICT)*, 2020, pp. 240--245.
- [16] S. Sadiq, M. Orłowska, W. Sadiq, and J. Lin, “Sqlator: An online sql learning workbench,” *SIGCSE Bull.*, vol. 36, no. 3, p. 223--227, jun 2004. [Online]. Available: <https://doi.org/10.1145/1026487.1008055>
- [17] B. Chandra, M. Joseph, B. Radhakrishnan, S. Acharya, and S. Sudarshan, “Partial marking for automated grading of sql queries,” *Proc. VLDB Endow.*, vol. 9, no. 13, p. 1541--1544, sep 2016. [Online]. Available: <https://doi.org/10.14778/3007263.3007304>
- [18] B. Chandra, A. Banerjee, U. Hazra, M. Joseph, and S. Sudarshan, “Automated grading of sql queries,” in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019, pp. 1630--1633.
- [19] I. Štajduhar and G. Mauša, “Using string similarity metrics for automated grading of sql statements,” in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015, pp. 1250--1255.
- [20] S. Brown and P. Knight, *Assessing Learners in Higher Education*, ser. Teaching and learning in higher education. Kogan Page, 1994. [Online]. Available: https://books.google.com/books?id=DX2t7Ck__3gC
- [21] G. Russell and A. Cumming, “Improving the student learning experience for sql using automatic marking.” 01 2004, pp. 281--288.
- [22] S. Brass and C. Goldberg, “Semantic errors in sql queries: a quite complete list,” in *Fourth International Conference on Quality Software, 2004. QSIC 2004. Proceedings.*, 2004, pp. 250--257.
- [23] R. B. Buitendijk, “Logical errors in database sql retrieval queries,” *Comput. Sci. Econ. Manage.*, vol. 1, no. 2, p. 79--96, jun 1988. [Online]. Available: <https://doi.org/10.1007/BF00427157>

- [24] T. Taipalus and P. Perälä, “What to expect and what to focus on in sql query teaching,” in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 198–203. [Online]. Available: <https://doi.org/10.1145/3287324.3287359>
- [25] T. Taipalus, “Explaining causes behind sql query formulation errors,” in *2020 IEEE Frontiers in Education Conference (FIE)*, 2020, pp. 1–9.
- [26] “Canvas, “what is canvas?,” what is canvas? - instructure community.” [Online]. Available: <https://community.canvaslms.com/t5/Canvas-Basics-Guide/What-is-Canvas/ta-p/45>. [Accessed:17-Apr-2023]
- [27] W. Z. Zhang and D. H. Holland, “Using containers to execute sql queries in a cloud,” in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018, pp. 26–27.
- [28] W. Lehner and K.-U. Sattler, “Database as a service (dbaas),” in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 2010, pp. 1216–1217.
- [29] K. Atcharyachanvanich, S. Nalintippayawong, and T. Permpool, “Development of a mysql sandbox for processing sql statements: Case of dml and ddl statements,” in *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2017, pp. 1–6.
- [30] “Mysql :: Mysql 8.0 reference manual :: 2.2 installing mysql on unix/linux using generic binaries.” [Online]. Available: <https://dev.mysql.com/doc/refman/8.0/en/binary-installation.html>. [Accessed:10-Nov-2023]
- [31] “Postgresql: The world’s most advanced open source database.” [Online]. Available: <https://www.postgresql.org/>. [Accessed:10-Nov-2023]
- [32] “Database | oracle.” [Online]. Available: <https://www.oracle.com/database/>. [Accessed:10-Nov-2023]