Automating Database Question Generation and Marking with PrairieLearn

Final Document

Version:
First Version - July 28th, 2023
Final Version - August 15th, 2023

Contacts:
Matthew Obirek (matthewobirek@gmail.com),
Skyler Alderson (skyler@thealdersons.org),
Andrei Zipis (Andrei_Zipis@hotmail.com),
Nishant Srinivasan (nishant.srinivasan236@gmail.com)

Table of Contents

        3.1.2. RelaX Grading
        3.1.3. RelaX Autogeneration
    3.2. SQL
        3.2.1. SQL Static Loading
        3.2.2. SQL Grading
        3.2.3. SQL Autogeneration
4. Takeaways

Section 5: Product Delivery

1. Code Location
2. RelaxQueryApi Location
3. Deploy Location
4. Development Installation Dependencies
5. Development Instructions
6. Deployment Installation Dependencies
7. Deployment Instructions
8. Testing Instructions
9. Testing Not Implemented
10. Features not Implemented
11. Future Work
12. Workflow Statistics
    a. Clockify Hours
    b. Kanban Board
    c. Burnup Chart
13. Known Bugs
14. Core Requirements Delivered
15. Lessons Learned

Section 1: Scope and Charter

1. Identification

Category	Information
Project	Automating Database Question Generation and Marking with PrairieLearn
Purpose	Centralize all tools used in the first three labs of COSC 304 - Introduction to Databases by automating relational algebra and SQL question delivery and evaluation using PrairieLearn
Client	Dr. Ramon Lawrence
Project Sponsor	Dr. Ramon Lawrence
Project Manager	Nishant Srinivasan
Institution	University of British Columbia - Okanagan (UBC-O)

2. Project Purpose

As of right now, students in COSC 304 are reliant on using multiple different platforms for their coursework - including time-sensitive evaluations such as midterms and final examinations. These tools include Canvas, PrairieLearn, RelaX, GitHub, and a DBMS software to run and verify their queries locally.

The client would like to centralize all these tools in PrairieLearn for the students' convenience and relieve them of untimely technical issues that may arise with using any or all of these tools at the same time.

3. Project Scope, Objectives, and Success Criteria

3.1. Objectives and Success Criteria

Our objective is to integrate questions from specific, existing labs for COSC 304 from Canvas and GitHub into PrairieLearn, integrate the necessary tools for these labs such as RelaX, and automate randomized question generation as well as their evaluation process. In addition, the project requires us to implement a feature that allows students to verify their answers on PrairieLearn so that they need not rely on the DBMS software.

The team will add Python question generation and evaluation scripts for the specified labs and make front-end changes to allow usage of these scripts. The team will modify the user-interface by integrating RelaX into PrairieLearn and also implement a button that lets students verify their solutions without submitting.

3.2. Out of Scope

PrairieLearn is a pre-existing system with many features that are required for our system to function correctly. The following use cases are critical for our system to run but have been previously developed and are outside the scope of this project:

Existing Student Use Requirements

- Users will register to the PrairieLearn system via CWL login.
- Users will login to the PrairieLearn system via CWL login.
- Users will be able to view assessments, questions, grades.
- Users will be able to submit their answers.
- Users will be able to view submitted answers.
- Users will receive grades for submitted answers.

Existing Professor User Requirements

- Users will login to the PrairieLearn system.

- Users will select question types
- Users will be able to see students' submissions
- Users will be able to see and edit students' grades.
- Users will be able to see, edit, and mark students' individual questions.
- Users will be able to set time limits.
- Users will be able to set the number of allowed submissions.

4. High Level Requirements

- Students enrolled in COSC 304 at UBC-O will be able to complete labs 1, 2, and 3 entirely through PrairieLearn.
- All problems pertaining to the first three COSC 304 labs will be automatically marked once submitted.
- Submitting to PrairieLearn will be scalable with a large COSC 304 class size so that the service is not interrupted and performance does not degrade below acceptable levels.
- The user interface of the PrairieLearn questions will mimic the interface required to complete the first three labs.
    - RelaX will be ported for Lab 1
    - A textbox for DDL/SQL will be provided for Labs 2 & 3
        - UI will mimic SQuirrelSQL

5. Assumptions

- Team is able to code in Python.
- Team is able to code in JavaScript / Node.js.
- Team is able to code in HTML and CSS.
- Team is able to do front-end/back-end web development.
- Team is able to work with Databases.
- Team is able to use Docker.
- Team maintains consistent contact with the client and provides them with weekly updates.
- Timely and weekly scheduled meetings with client.
- Team clearly communicates with the client about any issues and/or questions.
- Questions for specified labs are integrated into PrairieLearn successfully.
- RelaX is integrated into the PrairieLearn UI for appropriate labs.
- Instructors are able to successfully automatically generate problems from specified labs.
- Students are able to verify accuracy of queries before submitting.
- Team meets deadlines.

6. High Level Project Description

Deliverable Description

- The deliverable will be an instance of PrairieLearn – software that allows for complex questioning and answering over the internet – hosted on local servers at UBC Okanagan Campus. The PrairieLearn instance will be able to automatically answer questions from COSC 304 lab 1 - Relational Algebra, COSC 304 lab 2 - SQL and MySQL table creation, and COSC 304 lab 3 – SQL and MySQL table querying.

Elements to be Implemented

- For lab 1 - Relational Algebra, The RelaX user interface must be implemented, and an automatic marking of the string used.
- For lab 2 - SQL and MySQL table creation, a live instance of SQL must be query-able before the answer is sent for automatic marking.
- For lab 3 - SQL and MySQL table querying, a live instance of SQL must be query-able before the answer is sent for automatic marking.
- For all three labs, questions should be generalized - abstracted - for easy automated generation.
- For all three labs, as well as the previous PrairieLearn projects, local hosting at UBC Okanagan must be achieved.

7. High Level Risks

- One or more team member(s) become unavailable.
    - Sickness.
    - Personal reasons.
    - Bereavement.
- One or more team member(s) refuse to work with the group or on the project.
- One or more team member(s) exaggerated capabilities.
- Miscommunication between client and team.
- Early errors not discovered until late in the project.
- External events increase stress for all stakeholders.
    - A pandemic occurs.
    - A natural disaster occurs.
- Project is delivered with unknown issues resulting in an unusable product.
- PrairieLearn or RelaX are removed from GitHub prior to project completion.

8. Stakeholder List

- Developers - Our Team
    - Project Manager: Nishant Srinivasan
    - Client Liaison: Matthew Obirek
    - Technical Lead: Skyler Alderson
    - Integration Lead: Andrei Zipis
- Clients
    - UBC Okanagan Campus
    - Dr Ramon Lawrence
- Maintainers
    - Dr.Ramon Lawrence

    - Future contributors/developers
- Users
    - University Professors
        - COSC 304,
        - Potentially COSC 111, 121
    - University Students
        - COSC 304,
        - Potentially COSC 111, 121

9. Major Milestones

1. May 29th, Week 3: First draft of charter, scope, and requirements documents.
2. June 4th, Week 4: Final - Charter, scope, requirements, and preliminary design documents.
3. June 9th, Week 4: Final - Design Documents.
4. July 2nd, Week 8: Minimum viable product presentation.
5. August 13th, Week 14: Completion of code, final report.

10. Budget

This project has a monetary budget of $0.

This project has a time budget of 64 to 80 people hours per week for fourteen weeks. It has an approximate total time budget of 900 hours for a team of four developers.

11. Owner, Project Manager, Developers

- Owner: Ramon Lawrence
- Project Manager: Nishant Srinivasan
- Client Liaison: Matthew Obirek
- Technical Lead: Skyler Alderson
- Integration Lead: Andrei Zipis

12. Problem Statement

The web service PrairieLearn does not currently support the ability for users to test SQL or relational algebra queries prior to submission. The automatic generation and marking of questions for SQL and relational algebra is not available to users on PrairieLearn.

13. Requirements

13.1. Functional Requirements

1. Provide documentation on how to deploy PrairieLearn on docker.
2. System will allow for relational algebra statements to be entered.
3. System will show visualizations of the resulting entered statement prior to submission.
4. System will automatically mark the relational algebra questions once submitted.
5. System will allow for DDL/SQL code to be entered.
6. System will show resulting tables of queries prior to submission.
7. System will automatically mark the DDL/SQL questions once submitted.
8. Student will be able to see the correct answer if the professor has allowed for the correct answer to be displayed after the question is submitted.
9. Professor will be able to set whether the correct answer will be displayed after the question is submitted.
10. Professor will be able to see the correct answer.

13.2. Non-Functional Requirements

1. PrairieLearn will be deployed with Docker.
2. The system will support all COSC 304 users simultaneously – about 200 students.
3. The system will ensure data integrity and preservation so that no data is lost upon submission.
4. The system will display entered queries within 3 seconds at scale and under optimal conditions.
5. The system will return automarked submissions within 5 seconds at scale and under optimal conditions.
6. The user interface will match existing software used for COSC 304.
7. The software will be maintainable. Code will be modular and appropriately commented.

13.3. Technical Requirements

1. Rebuild RelaX editor and calculator into PrairieLearn.
2. Frontend: JavaScript, HTML, CSS.
3. Backend: Python, Node.JS.
4. Write JavaScript code that takes in SQL/DDL statements and displays appropriate table results.
5. Write Python code that automatically marks submitted data and returns the students grade.
6. Version control will be established with Github
7. CI/CD pipelines will be established using GitHub Actions & DroneCI to automate testing and deployment.

13.4. User Requirements

1. Users will be able to enter relational algebra statements.
2. Users will be able to enter SQL statements.

3. Users will receive visual feedback on relational algebra statements prior to submission.
4. Users will receive visual feedback on SQL statements prior to submission

13.5. Stretch Goals

1. System gives partial marks for submitted answers.

14. Project Deliverables

1. Charter, Scope, Requirements, Planning documents
    a. Use cases
    b. Workflow
    c. Work breakdown structure
2. RelaX integration, question randomization, and auto marking
3. DDL integration, question randomization, and auto marking
4. SQL integration, question randomization, and auto marking

15. Methodology / Workflow

Our project will follow an agile methodology with a test-driven development (TDD) approach. All tests will be written before implementing each feature, ensuring that they first fail. We will be utilizing the Kanban framework through a Kanban board on GitHub projects to manage and visualize our workflow. Time tracking of tasks will be done using Clockify. The integrated development environment (IDE) we have chosen to use for both our front-end and back-end is VSCode. This is due to our familiarity with it and the many extensions it has allowing for a variety of programming languages and integration with GitHub and Docker. We will be using DroneCI for continuous integration (CI).

The group will be divided into two groups of two individuals per feature. This is due to the different requirements of integrating RelaX for Lab 1 and SQL/DDL for Labs 2 & 3. This division aims to optimize resource allocation while ensuring scrutiny and attention to detail for each feature.

16. Use Cases

These use cases build upon existing PrairieLearn documentation and only include items modified by this project. They should not be taken as a complete set of design documents.

16.1. Student Use Cases Diagram

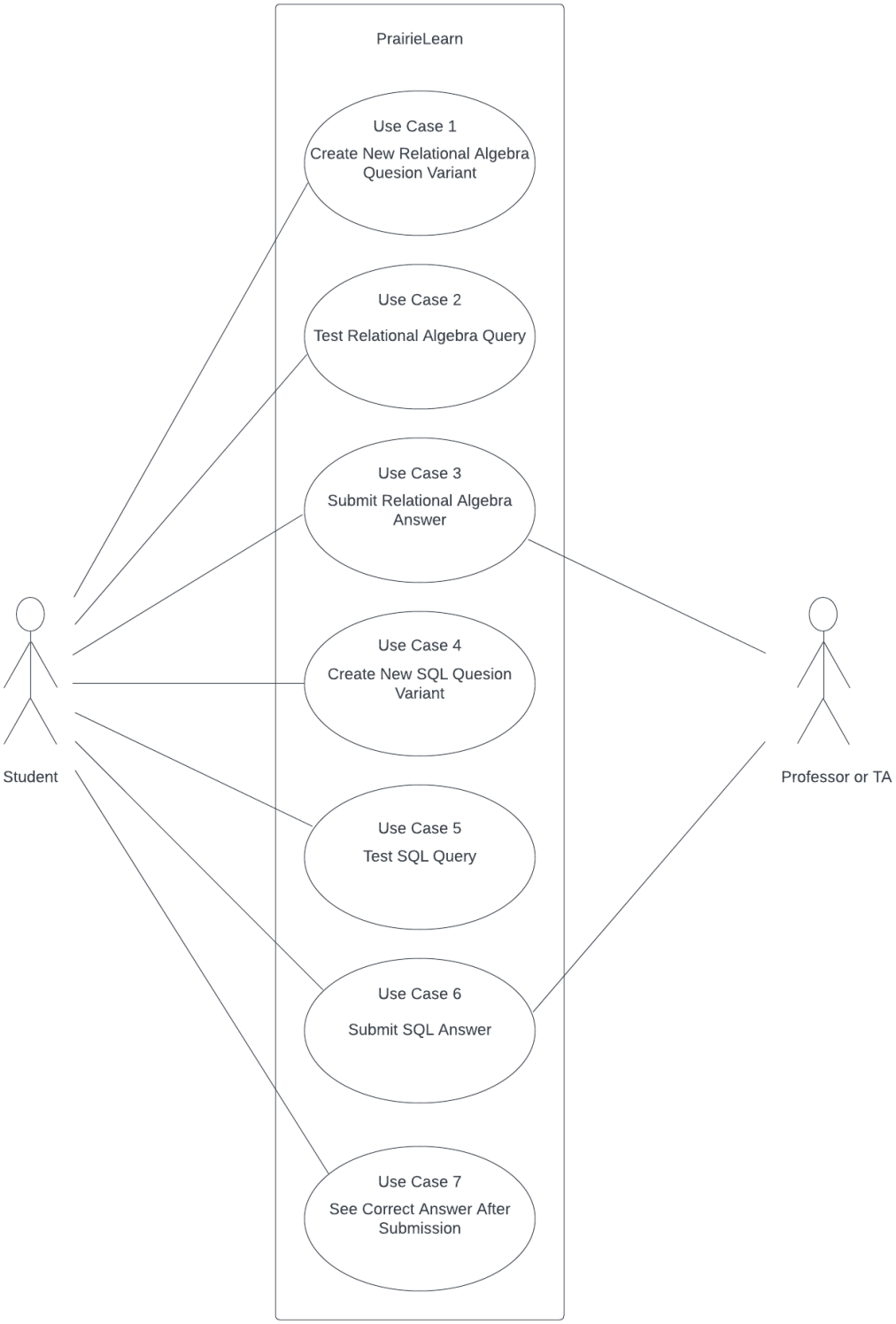


Figure 16.1: *A student may interact with the system (PrairieLearn) either through a Relation Algebra Lab or Answer SQL Lab. In either lab, the student may generate new question variants, test queries, and submit an answer. The professor or TA are able to see the student's grades after they have submitted their answer. After the final submission, a student may see the correct answer depending on the question's answer visibility settings.*

16.2. Use Case 1: Create New Relational Algebra Variant

ID: 1
Primary actor: Student
Description: A student wishes to obtain a new question variant
Precondition: Student has successfully logged in to PrairieLearn and selected a relation algebra question.
Postcondition: If the student has successfully accessed the question and generates a new question variant, the student sees a new variant of the relational algebra question.

Main Scenario:

1. Student selects the relation algebra question.
2. Student selects "Generate New Variant".

Extensions:

None

16.3. Use Case 2: Test Relational Algebra Query

ID: 2
Primary actor: Student
Description: A student wishes to receive feedback on their answer prior to submission.
Precondition: Student has successfully logged in to PrairieLearn and selected the relation algebra question.
Postcondition: If the student successfully accessed the lab and formats their question, the student will see the results of their query.

Main Scenario:

1. Student selects the relation algebra question.
2. Student enters their answer in the RelaX editor.
3. Students tests their query without submission.
4. The output of the query is displayed for the student.

Extensions:

3a. The student's answer includes one or more formatting errors.
    3a1. System issues an error message, informing the student of the error and

prevents the query from being run.

16.4. Use Case 3: Submit Relational Algebra Answer

ID: 3
Primary actor: Student
Description: A student wishes to submit their answer for grading.
Precondition: Student has successfully logged in to PrairieLearn and selected the relation algebra question.
Postcondition: If the student successfully accessed the lab and formats their question, the student will receive feedback on their answer.

Main Scenario:

1. Student selects the relation algebra question.
2. Student enters their answer in the RelaX editor.
3. Student submits their answer.
4. Student receives feedback on their answer.

Extensions:

3a. The student's answer includes one or more formatting errors.
    3a1. System issues an error message, informing the student of the error and prevents the query from being run.
3b. The student's answer is incorrect and the student has remaining attempts.
    3b1. The student is informed their answer is incorrect and is prompted to try again.
3c. The student's answer is incorrect and the student has no remaining attempts.
    3c1. The student is informed their answer is incorrect.

16.5. Use Case 4: Create New SQL Variant

ID: 4
Primary actor: Student
Description: A student wishes to obtain a new question variant
Precondition: Student has successfully logged in to PrairieLearn and selected an SQL question.
Postcondition: If the student has successfully accessed the question and generates a new question variant, the student sees a new variant of the SQL question.

Main Scenario:

1. Student selects the SQL question.
2. Student selects "Generate New Variant".

Extensions:

None

16.6. Use Case 5: Test SQL Query

ID: 5
Primary actor: Student
Description: A student wishes to receive feedback on their answer prior to submission.
Precondition: Student has successfully logged in to PrairieLearn and selected the SQL question.
Postcondition: If the student successfully accessed the lab and formats their question, the student will see the results of their query.

Main Scenario:

1. Student selects the SQL question.
2. Student enters their answer in the SQL editor.
3. Students tests their query without submission.
4. The output of the query is displayed for the student.

Extensions:

3a. The student's answer includes one or more formatting errors.
    3a1. System issues an error message, informing the student of the error and prevents the query from being run.

16.7. Use Case 6: Submit SQL Answer

ID: 6
Primary actor: Student
Description: A student wishes to submit their answer for grading.
Precondition: Student has successfully logged in to PrairieLearn and selected the SQL question.
Postcondition: If the student successfully accessed the lab and formats their question, the student will receive feedback on their answer.

Main Scenario:

1. Student selects the SQL question.
2. Student enters their answer in the SQL editor.
3. Student submits their answer.
4. Student receives feedback on their answer.

Extensions:

3a. The student's answer includes one or more formatting errors.
    3a1. System issues an error message, informing the student of the error and prevents the query from being run.
3b. The student's answer is incorrect and the student has remaining attempts.
    3b1. The student is informed their answer is incorrect and is prompted to try again.
3c. The student's answer is incorrect and the student has no remaining attempts.
    3c1. The student is informed their answer is incorrect.

16.8. Use Case 7: See Correct Answer After Submission

ID 7:
Primary actor: Student
Description: A student wishes to complete their question to see if it is correct.
Precondition: Student has successfully logged in to PrairieLearn and selected the desired question.
Post condition: If the student successfully accessed the lab, submitted the question, and the professor has set the question's solution to be visible, then the question will be marked and the student will see the correct answer.

Main Scenario:

1. Student selects the desired question.
2. Student enters their answer.
3. Student submits their answer.
4. Student receives feedback on their answer and is able to see the correct solution.

Extensions:

3a. The student's answer includes one or more formatting errors.
    3a1. System issues an error message, informing the student of the error and prevents the query from being run.
3b. The student's answer is incorrect and the student has remaining attempts.
    3b1. The student is informed their answer is incorrect and is prompted to try again.
3c. The student's answer is incorrect and the student has no remaining attempts.
    3c1. The student is informed their answer is incorrect.

16.9. Professor Use Case Diagram

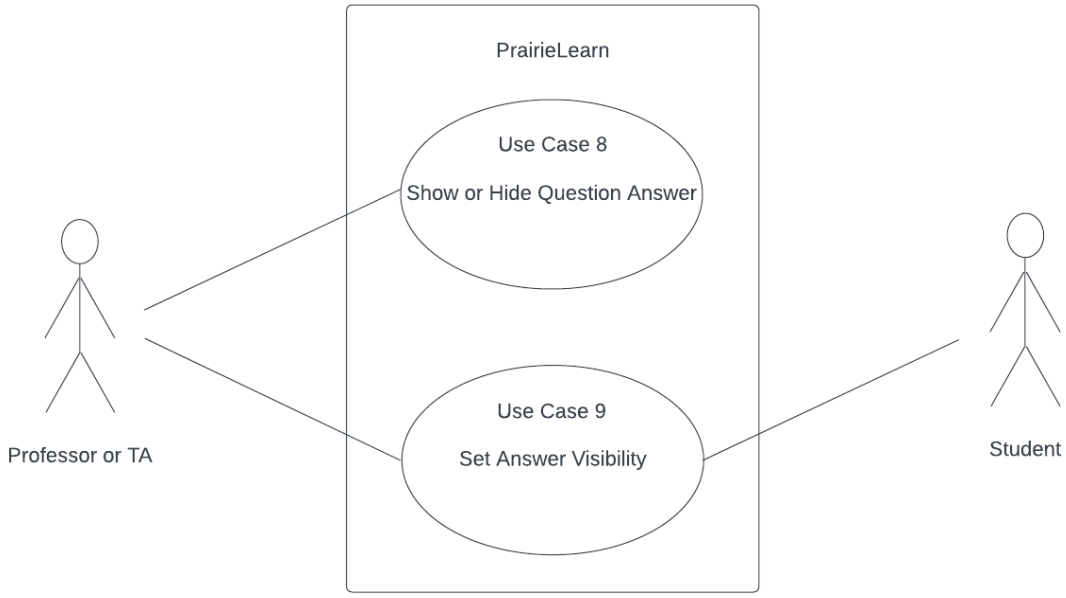


Figure 16.9: *A professor or TA is able to interact with the system (PrairieLearn) either by viewing a question's answer or by setting the visibility of a solution after the student submits an answer.*

16.10. Use Case 8: Show or Hide Answer

ID: 8
Primary actor: Professor
Description: A professor wishes to view the correct answer to a desired question.
Precondition: Professor has successfully logged into PrairieLearn and selected the desired question.
Post Condition: If the professor successfully accessed the lab, then the professor sees the question solution.

Main Scenario:

1. Professor selects the desired question.
2. Professor selects "Show/Hide Answer".

Extensions:

None

16.11. Use Case 9: Set Answer Visibility

ID: 9

Primary actor: Professor
Description: A professor wishes to change whether the answer is viewable after the question is answered by a student.
Precondition: Professor has successfully logged into PrairieLearn and selected the desired question.
Post Condition: If the professor has successfully logged into PrairieLearn, then the question visibility is set.

Main Scenario:

1. Professor selects the desired question.
2. Professor navigates to the “Files” tab.
3. Professor selects “info.json”.
4. Professor sets desired visibility.

Extensions:

None

17. Work Breakdown Structure

Tasks	Nishant		Andrei		Skyler		Matthew		Average Hours	
	Estimate	**Actual**	**Estimate**	**Actual**	**Estimate**	**Actual**	**Estimate**	**Actual**	**Estimate**	**Actual**
Assignments									**21.5**	**24**
Scope, Charter, & WBS	5	3.5	5	7	5	5	4	4	4.75	4.875
Design Documents	8	3.5	4	7	10	15.5	1	0	5.75	6.5
MVP Review and Prep.	6	5	6	5	5	5	4	2	5.25	4.25
Final Review and Prep.	6	12.5	6	10	7	6	4	5	5.75	8.375
Meetings									**52**	**60.25**
Team	50	46.5	30	50	30	54	30	40	35	47.625
Client	10	5.5	10	5	10	10	10	6	10	6.625
Reporting	7	5	7	7	7	5	7	7	7	6
Logs									**9.5**	**8.375**

Team	0	0	0	0	5	3	0	0	1.25	0.75
Dashboards	2	3	2	3	2	1	2	0	2	1.75
Individual	5	3.5	5	5	5	3	5	7	5	4.625
Communications	0	0	0	1	0	0	5	4	1.25	1.25
Environment Setup									**21.625**	**8.25**
Finding Linter	1	0	1	0	3	2	1	0	1.5	0.5
Installing and Setting Up IDE's	1.5	0	1	1	1	1.5	2	2	1.375	1.125
Docker :^(	15	12	15	8	15	1.5	30	5	18.75	6.625
Familiarization									**19.25**	**7.5**
AutoER Documentation	4	0	5	3	3	1	5	2	4.25	1.5
Docker Deployment + Setup	10	10	10	3	10	1	10	1	10	3.75
Investigate RelaX code	0	0	0	1	10	1	10	7	5	2.25
PrairieLearn Setup									**6.75**	**8.75**
PL Documentation	3	0	5	3	3	10	5	19	4	8
Deploying PL to Docker	3	0	3	3	2	0	3	0	2.75	0.75
Repo Management									**10.25**	**4.25**
Pull Requests	2	1	10	5	2	2	2.5	0.5	4.125	2.125
Branch Handling	1	1	1	2	0.5	1	2	1	1.125	1.25
Code Reviews	3	0.5	10	2	2	1	5	0	5	0.875
Documentation									**4.625**	**3.75**
Project Board	5	6	0.5	1	0.5	1		0	1.5	2
Updating Charter and Scope Concurrently	1	0.5	1	1	1	1	0.5	0	0.875	0.625
PL Docker	0.5	0	0.5	1	0.5	0	5	0.5	1.625	0.375

Deployment Documentation										
Instructor Guide Documentation	0.5	0	0.5	2	0.5	0	1	1	0.625	0.75
RelaX - Lab 1									**32.5**	**34.25**
Allow for Relational Algebra Statements to be Entered	0	0	0	0	20	0	30	44	12.5	11
Visualizations of the Resulting Entered Statement (Relation Algebra)	0	0	0	0	20	0	30	4	12.5	1
Automatically Mark the Relational Algebra Questions once Submitted.	0	0	0	30	10	0	10	0	5	7.5
Automatically Generate RelaX Questions.	0	0	0	0	10	2	0	57	2.5	14.75
SQL/DDL									**21.25**	**53.125**
Allow for DDL/SQL Code to be Entered.	20	8	20	26	0	0	0	0	10	8.5
Visualizations of Resulting tables	3	12.5	2	26	0	0	0	0	1.25	9.625
Automatically Mark the DDL/SQL Questions Upon Submission.	10	24	10	10	0	5	0	0	5	9.75
Automatically Generate DDL/SQL Questions.	10	0	10	0	0	101	0	0	5	25.25
Testing									**34.5**	**24.6875**
Mock Data Generation	4	0	4	0	4	0	2	0	3.5	0
UI Tests	10	8.75	10	3	5	0	5	0	7.5	2.9375

Unit Tests	18	20.25	20	11	20	5	10	1	17	9.3125
Usability Tests	2	5	2	3.5	8	0	2	0	3.5	2.125
Integration Tests	3	8.75	3	3	3	5	3	1	3	4.4375
Performance	0	4.5	0	1	0	0	0	0	0	1.375
Drone	0	15.5	0	1	0	1	0	0.5	0	4.5
Answer Visibility									**8**	**1.5**
Professor can set whether a question shows correct answer after student submits their answer	5	0	5	1	5	0	5	1	5	0.5
Professor can show/hide correct answer	1	0	1	1	1	0	1	1	1	0.5
Student is able to see correct answer after submitting their answer if the setting is set	2	0	2	1	3	0	1	1	2	0.5
Total	237.5	226.25	227.5	253.5	249	250.5	253	224.5	967	956

18. Approvals

_____

Project Sponsor (UBC Okanagan)

_____

Project Manager

Section 2: Design Documentation

1. Project Description

The project - “Automating Database Question Generation and Marking with PrairieLearn", is to successfully automate the delivery and evaluation for relational algebra and SQL questions on PrairieLearn. The purpose is to improve the delivery of COSC 304 for students at UBC-O for our client, Dr. Lawrence.

As of right now, students in COSC 304 are reliant on using multiple different platforms for their coursework - including time-sensitive evaluations such as midterms and final examinations. These tools include Canvas, PrairieLearn, RelaX, GitHub, and a DBMS software to run and verify their queries locally.

Our objective is to integrate questions from specific, existing labs for COSC 304 from Canvas and GitHub into PrairieLearn, integrate the necessary tools for these labs such as RelaX, and automate randomized question generation as well as their evaluation process. In addition, the project requires us to implement a feature that allows students to verify their answers on PrairieLearn so that they need not rely on the DBMS software.

2. User Groups and Personas

2.1. Students

Figure 2.1: *A persona for a student user profile named Sally Student.*

2.2. Professor

Figure 2.2: *A persona for a professor user profile named Travis Teacher.*

3. Use Cases

These use cases build upon existing PrairieLearn documentation and only include items modified by this project. They should not be taken as a complete set of design documents.

3.1. Student Use Cases Diagram

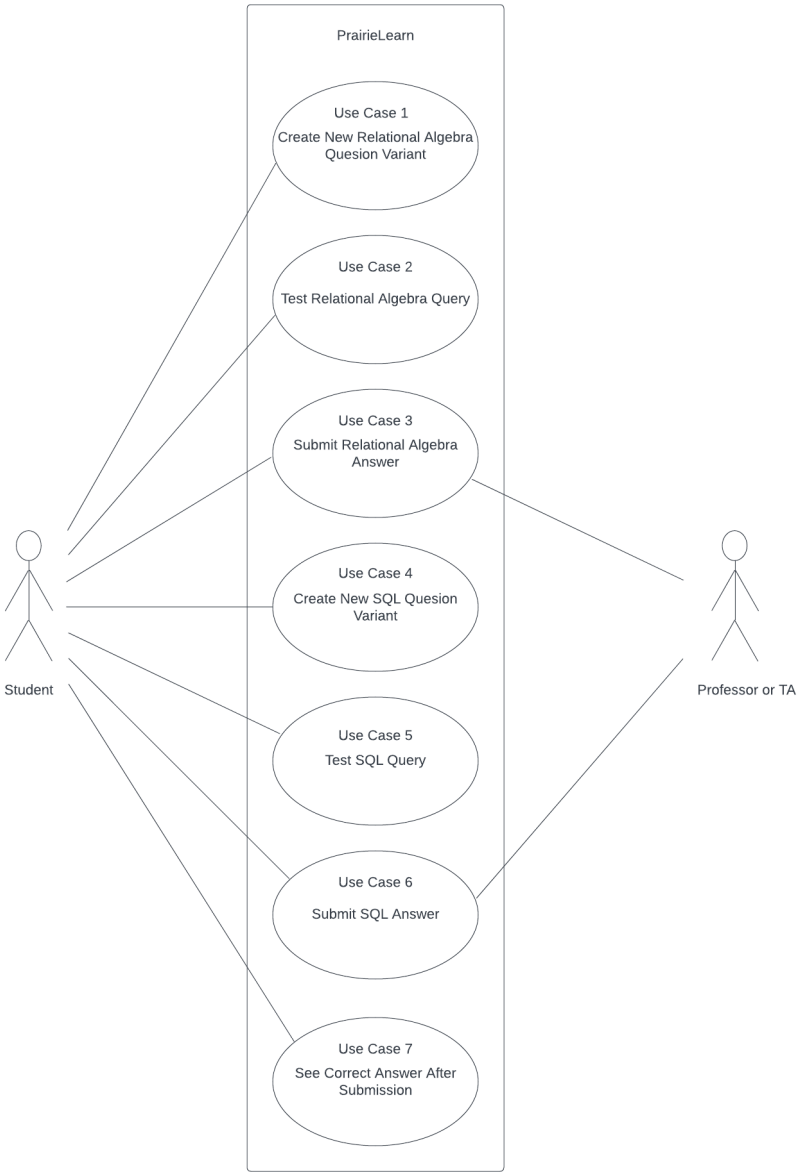


Figure 3.1: *A student may interact with the system (PrairieLearn) either through a Relational Algebra Lab or Answer SQL Lab. In either lab, the student may generate new question variants, test queries, and submit an answer. The professor or TA are able to see the student's grades after they have submitted their answer. After the final submission, a student may see the correct answer depending on the question's answer visibility settings.*

3.2. Use Case 1: Create New Relational Algebra Variant

ID: 1
Primary actor: Student
Description: A student wishes to obtain a new question variant
Precondition: Student has successfully logged in to PrairieLearn and selected a relational algebra question.
Postcondition: If the student has successfully accessed the question and generates a new question variant, the student sees a new variant of the relational algebra question.

Main Scenario:

1. Student selects the relational algebra question.
2. Student selects “Generate New Variant”.

Extensions:

None

3.3. Use Case 2: Test Relational Algebra Query

ID: 2
Primary actor: Student
Description: A student wishes to receive feedback on their answer prior to submission.
Precondition: Student has successfully logged in to PrairieLearn and selected the relational algebra question.
Postcondition: If the student successfully accessed the lab and formats their question, the student will see the results of their query.

Main Scenario:

1. Student selects the relational algebra question.
2. Student enters their answer in the RelaX editor.
3. Students tests their query without submission.
4. The output of the query is displayed for the student.

Extensions:

3a. The student’s answer includes one or more formatting errors.
    3a1. System issues an error message, informing the student of the error and prevents the query from being run.

3.4. Use Case 3: Submit Relational Algebra Answer

ID: 3
Primary actor: Student
Description: A student wishes to submit their answer for grading.

Precondition: Student has successfully logged in to PrairieLearn and selected the relational algebra question.
Postcondition: If the student successfully accessed the lab and formats their question, the student will receive feedback on their answer.

Main Scenario:

1. Student selects the relational algebra question.
2. Student enters their answer in the RelaX editor.
3. Student submits their answer.
4. Student receives feedback on their answer.

Extensions:

3a. The student's answer includes one or more formatting errors.
    3a1. System issues an error message, informing the student of the error and prevents the query from being run.
3b. The student's answer is incorrect and the student has remaining attempts.
    3b1. The student is informed their answer is incorrect and is prompted to try again.
3c. The student's answer is incorrect and the student has no remaining attempts.
    3c1. The student is informed their answer is incorrect.

3.5. Use Case 4: Create New SQL Variant

ID: 4
Primary actor: Student
Description: A student wishes to obtain a new question variant
Precondition: Student has successfully logged in to PrairieLearn and selected an SQL question.
Postcondition: If the student has successfully accessed the question and generates a new question variant, the student sees a new variant of the SQL question.

Main Scenario:

1. Student selects the SQL question.
2. Student selects "Generate New Variant".

Extensions:

None

3.6. Use Case 5: Test SQL Query

ID: 5
Primary actor: Student
Description: A student wishes to receive feedback on their answer prior to submission.
Precondition: Student has successfully logged in to PrairieLearn and selected the SQL question.
Postcondition: If the student successfully accessed the lab and formats their question, the student will see the results of their query.

Main Scenario:

1. Student selects the SQL question.
2. Student enters their answer in the SQL editor.
3. Students tests their query without submission.
4. The output of the query is displayed for the student.

Extensions:

3a. The student's answer includes one or more formatting errors.
    3a1. System issues an error message, informing the student of the error and prevents the query from being run.

3.7. Use Case 6: Submit SQL Answer

ID: 6
Primary actor: Student
Description: A student wishes to submit their answer for grading.
Precondition: Student has successfully logged in to PrairieLearn and selected the SQL question.
Postcondition: If the student successfully accessed the lab and formats their question, the student will receive feedback on their answer.

Main Scenario:

1. Student selects the SQL question.
2. Student enters their answer in the SQL editor.
3. Student submits their answer.
4. Student receives feedback on their answer.

Extensions:

3a. The student's answer includes one or more formatting errors.
    3a1. System issues an error message, informing the student of the error and prevents the query from being run.
3b. The student's answer is incorrect and the student has remaining attempts.
    3b1. The student is informed their answer is incorrect and is prompted to try again.
3c. The student's answer is incorrect and the student has no remaining attempts.
    3c1. The student is informed their answer is incorrect.

3.8. Use Case 7: See Correct Answer After Submission

ID 7:
Primary actor: Student
Description: A student wishes to complete their question to see if it is correct.
Precondition: Student has successfully logged in to PrairieLearn and selected the desired question.

Post condition: If the student successfully accessed the lab, submitted the question, and the professor has set the question's solution to be visible, then the question will be marked and the student will see the correct answer.

Main Scenario:

1. Student selects the desired question.
2. Student enters their answer.
3. Student submits their answer.
4. Student receives feedback on their answer and is able to see the correct solution.

Extensions:

3a. The student's answer includes one or more formatting errors.
  3a1. System issues an error message, informing the student of the error and prevents the query from being run.
3b. The student's answer is incorrect and the student has remaining attempts.
  3b1. The student is informed their answer is incorrect and is prompted to try again.
3c. The student's answer is incorrect and the student has no remaining attempts.
  3c1. The student is informed their answer is incorrect.

3.9. Professor or TA Use Case Diagram

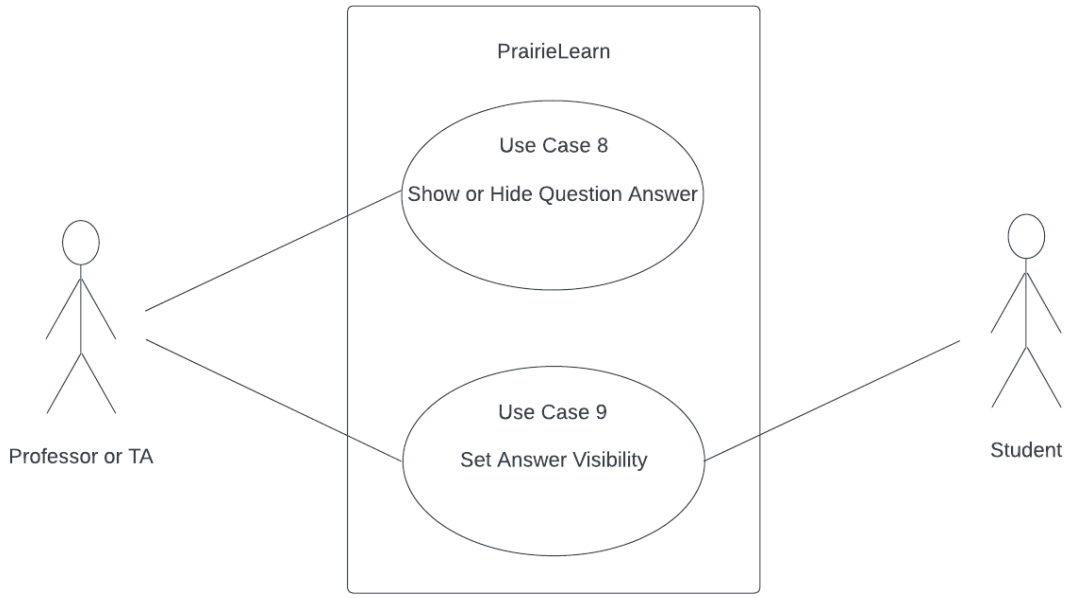


Figure 3.9: *A professor or TA is able to interact with the system (PrairieLearn) either by viewing a question's answer or by setting the visibility of a solution after the student submits an answer.*

3.10. Use Case 8: Show or Hide Answer

ID: 8
Primary actor: Professor
Description: A professor wishes to view the correct answer to a desired question.
Precondition: Professor has successfully logged into PrairieLearn and selected the desired question.
Post Condition: If the professor successfully accessed the lab, then the professor sees the question solution.

Main Scenario:

1. Professor selects the desired question.
2. Professor selects “Show/Hide Answer”.

Extensions:

None

3.11. Use Case 9: Set Answer Visibility

ID: 9
Primary actor: Professor
Description: A professor wishes to change whether the answer is viewable after the question is answered by a student.
Precondition: Professor has successfully logged into PrairieLearn and selected the desired question.
Post Condition: If the professor has successfully logged into PrairieLearn, then the question visibility is set.

Main Scenario:

1. Professor selects the desired question.
2. Professor navigates to the “Files” tab.
3. Professor selects “info.json”.
4. Professor sets desired visibility.

Extensions:

None

4. System Architecture

4.1. PrairieLearn Question Lifecycle

The following diagram (figure 4.1.1) displays the steps performed to create, grade, and otherwise handle a single relational algebra or SQL question.

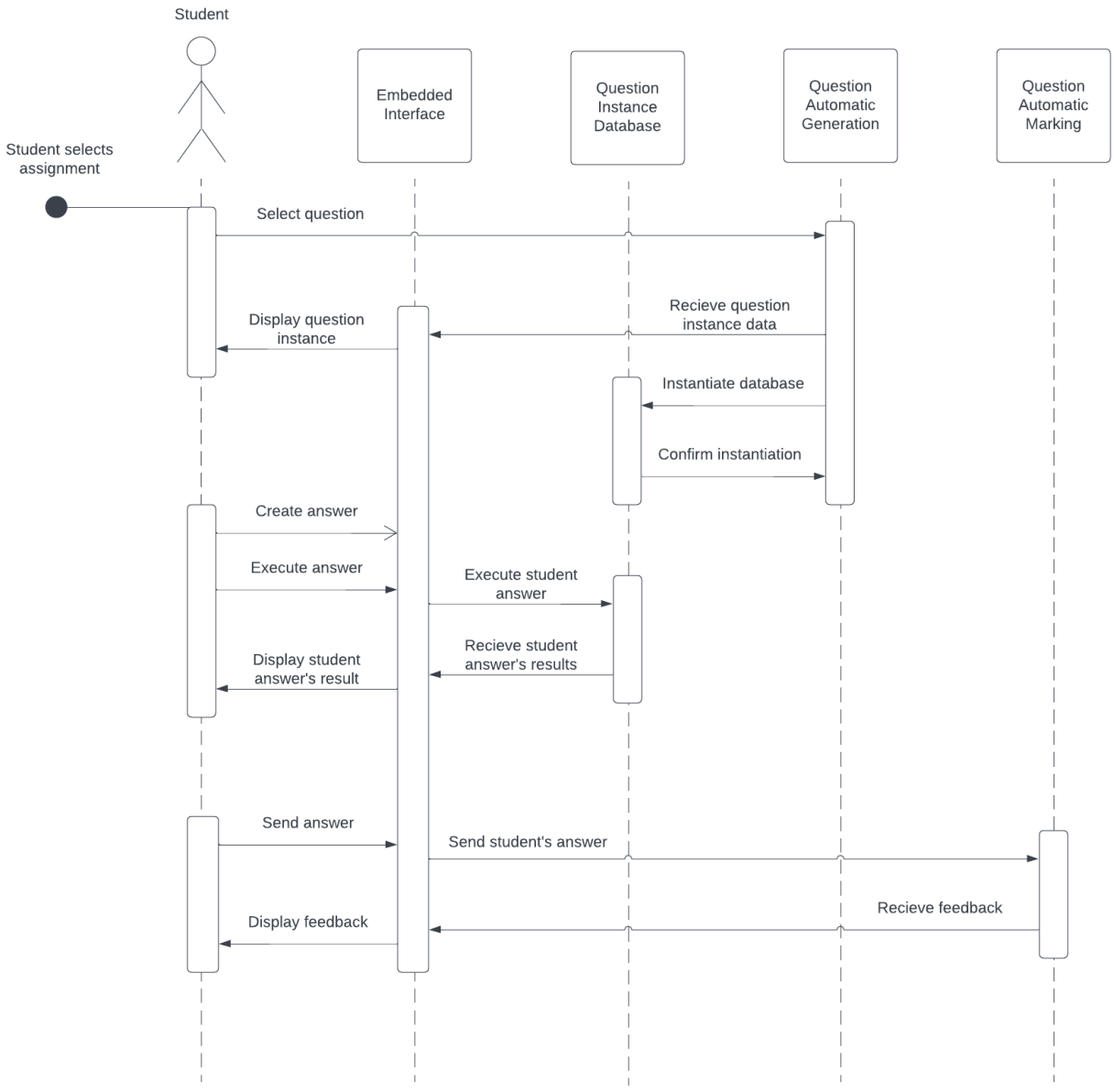


Figure 4.1.1: *After a student selects an assignment and a question, the question's parameters are sent to the question automatic generation. Question automatic generation creates the question instance which is used to both display the question to the user and to instantiate the question instance database. After the student creates their answer in the embedded interface, they execute the query which is sent to the question instance database and returns the query's results. These results are then displayed to the student. Finally, the student submits their query which is passed to question automatic grading which returns feedback to the student.*

4.2. Sequence Diagrams

4.2.1. Sequence Diagram for Use Cases 1 & 4

The following diagram (figure 4.2.1) shows the steps taken to generate a new question which is equivalent to generating a new question variant.

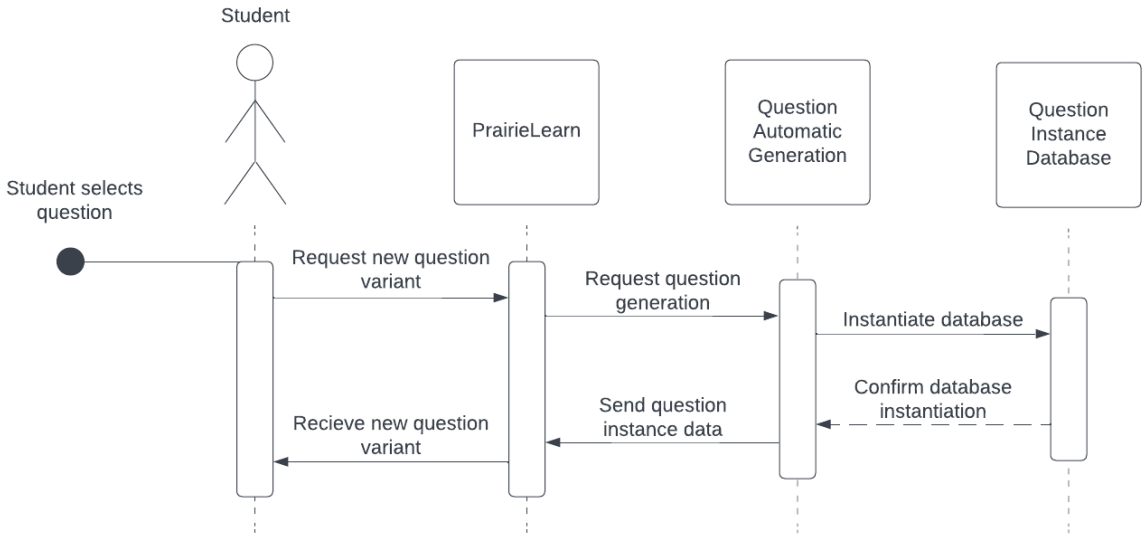


Figure 4.2.1: *After a student selects a question, they request a new question variant from the PrairieLearn system. PrairieLearn uses its automatic question generation to create a new question variant and in so doing instantiates a new database. The question instance data is returned to the student.*

4.2.2. Sequence Diagram for Use Cases 2 & 5

The following diagram (figure 4.2.2) shows the steps taken to execute the user's current query and display its results.

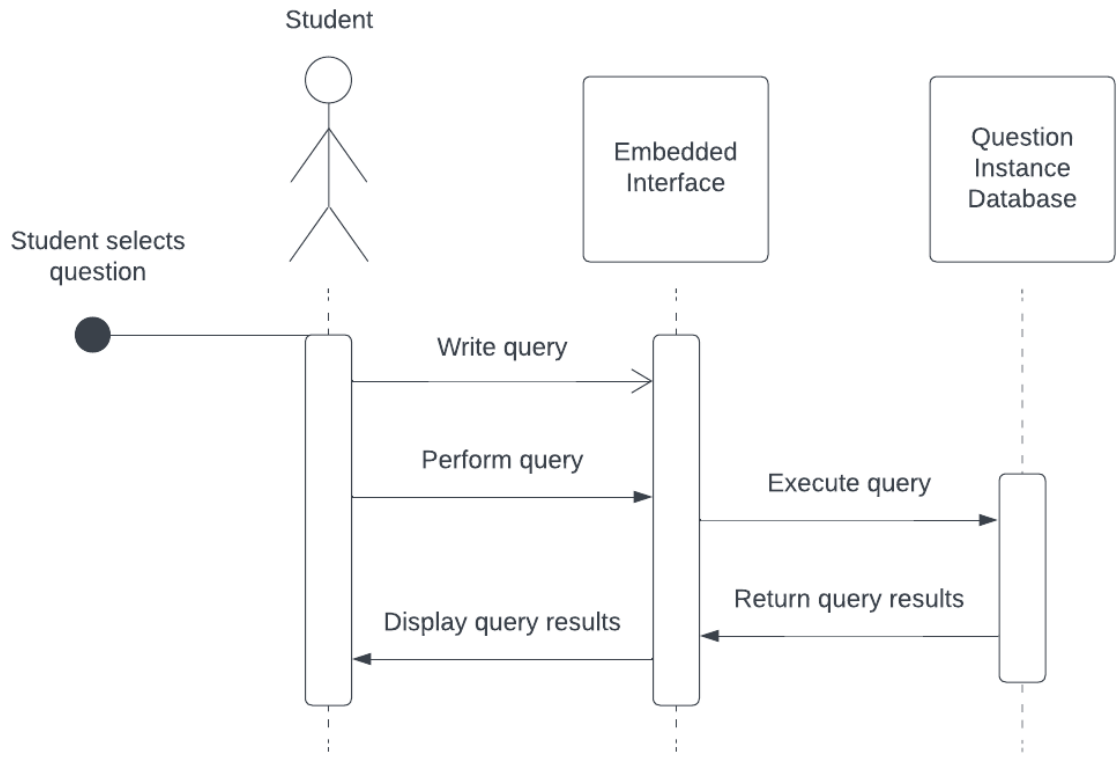


Figure 4.2.2: *After a student selects a question, they use the embedded editor to create a query. The student then tests the query which is run on the question instance database. The results of the query are returned to the student.*

4.2.3. Sequence Diagram for Use Cases 3, 6, & 7

The following diagram (figure 4.2.3) shows the steps taken to grade the student's answer and display feedback.

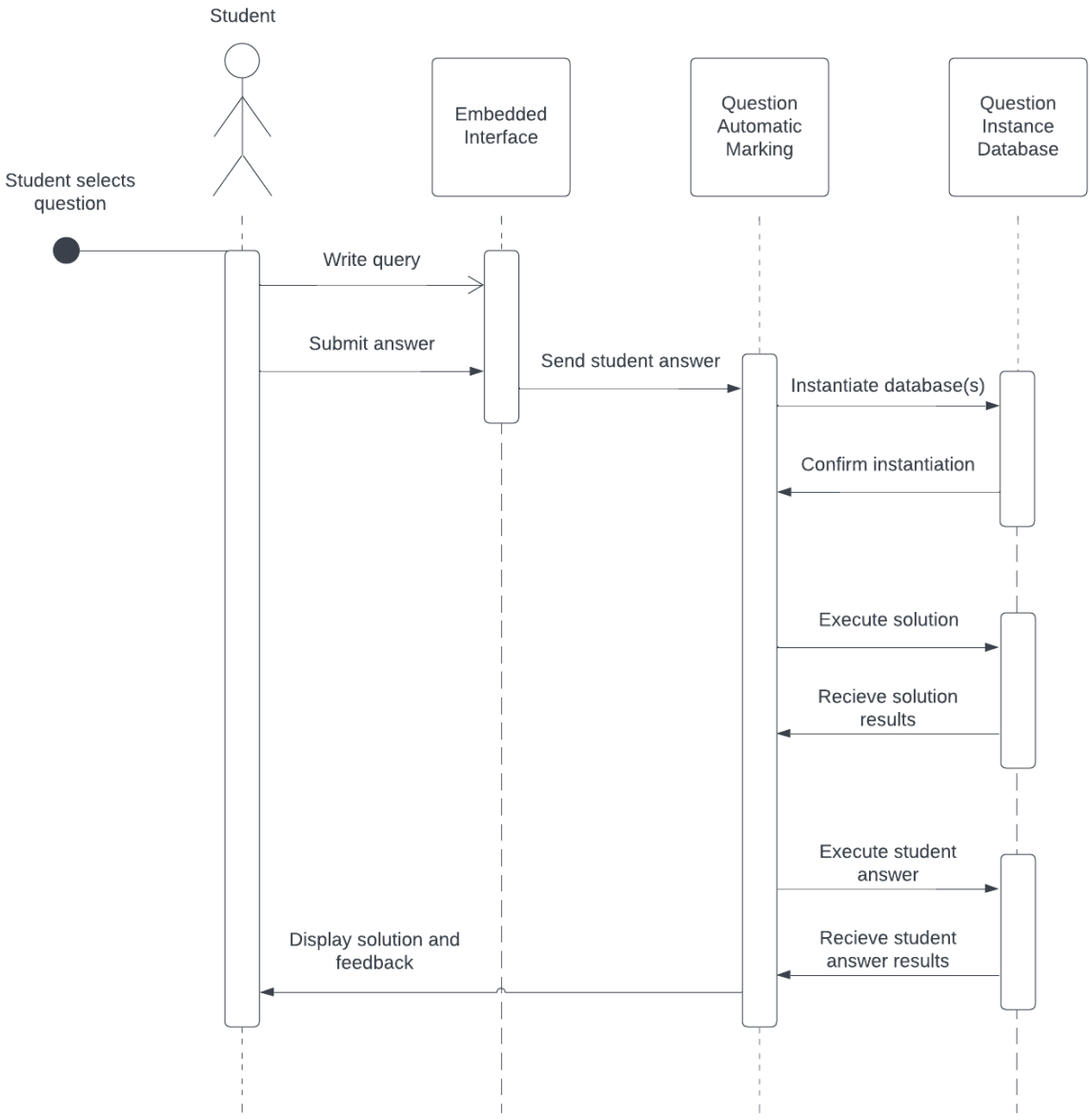


Figure 4.2.3: *After a student selects a question, they use the embedded editor to create a query. The student then submits their answer, where it is sent to question automatic marking. Question automatic marking then instantiates the backend question instance database and on it runs both the correct solution and*

the student's answer. Question automatic marking then compares the two query results, using it to return feedback to the student.

4.2.4. Sequence Diagram for Use Case 8

The following diagram (figure 4.2.4) shows the steps taken for a professor or TA to request and see a question's solution.

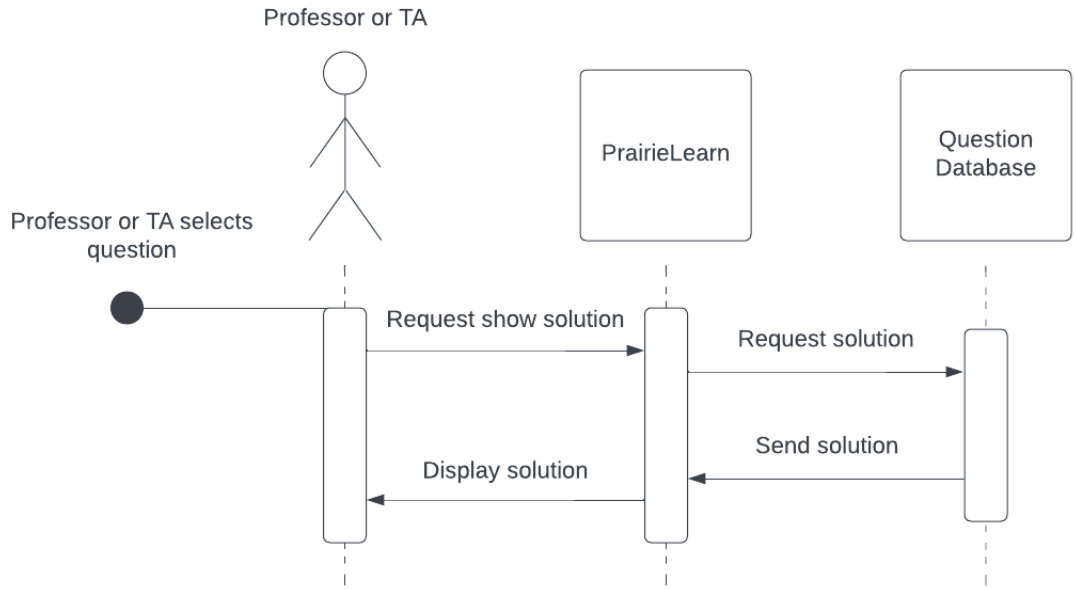


Figure 4.2.4: *After a professor or a TA selects a question, they request to see the answer. PrairieLearn then obtains the solution from the question database and shows it to the professor or TA.*

4.2.5. Sequence Diagram for Use Case 9

The following diagram (figure 4.2.5) shows the steps taken for a professor or TA to set whether or not a question will display the correct answer after the student's final submission.

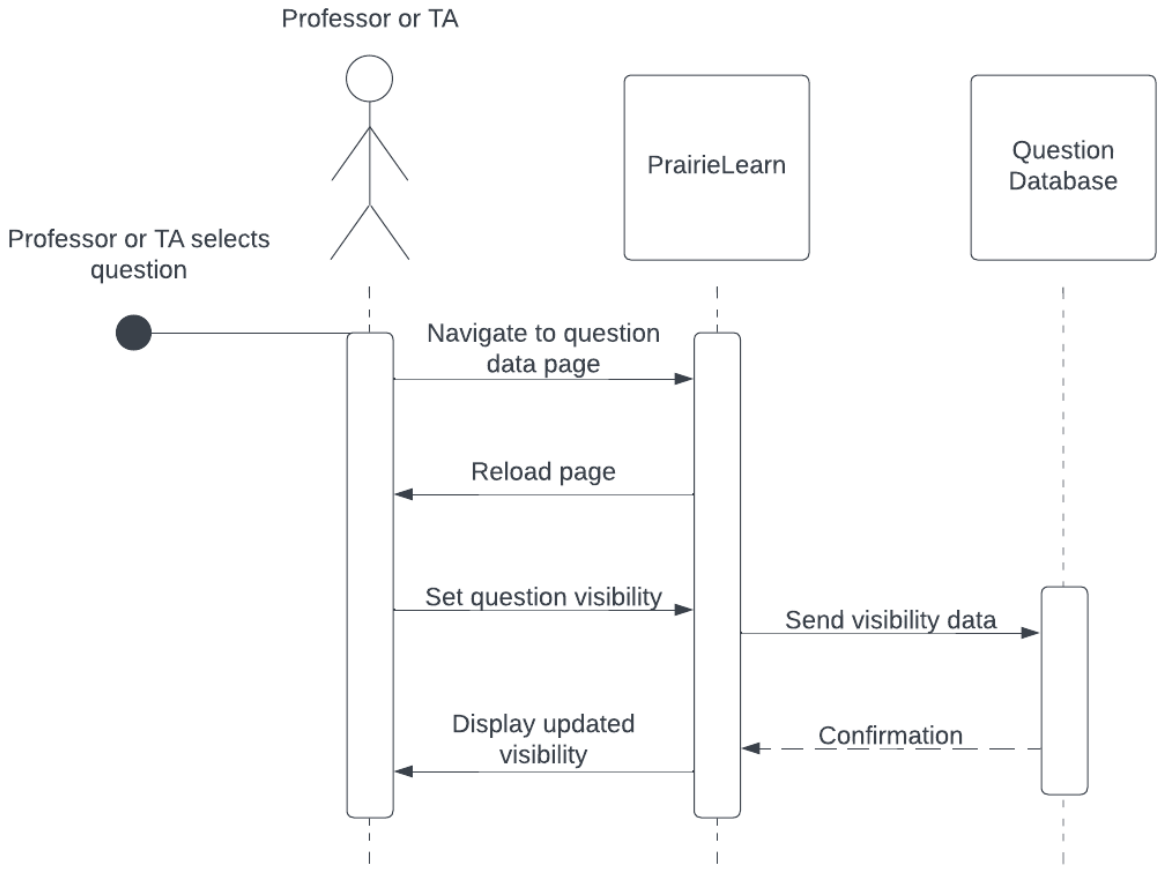


Figure 4.2.5: *After a professor or a TA selects a question, they navigate to the files tab of PrairieLearn and selects the question data. The professor or TA then updates the solution's visibility, where PrairieLearn changes the visibility settings and displays the updated status.*

4.3. Data flow Diagrams

4.3.1. Level 0 Data Flow Diagram

The following diagram (figure 4.3.1) shows how external actors (the student as well as both professors and TAs) interact with the systems (PrairieLearn, the embedded interface, and the RelaX Grader) as well as how information is transformed between each system.

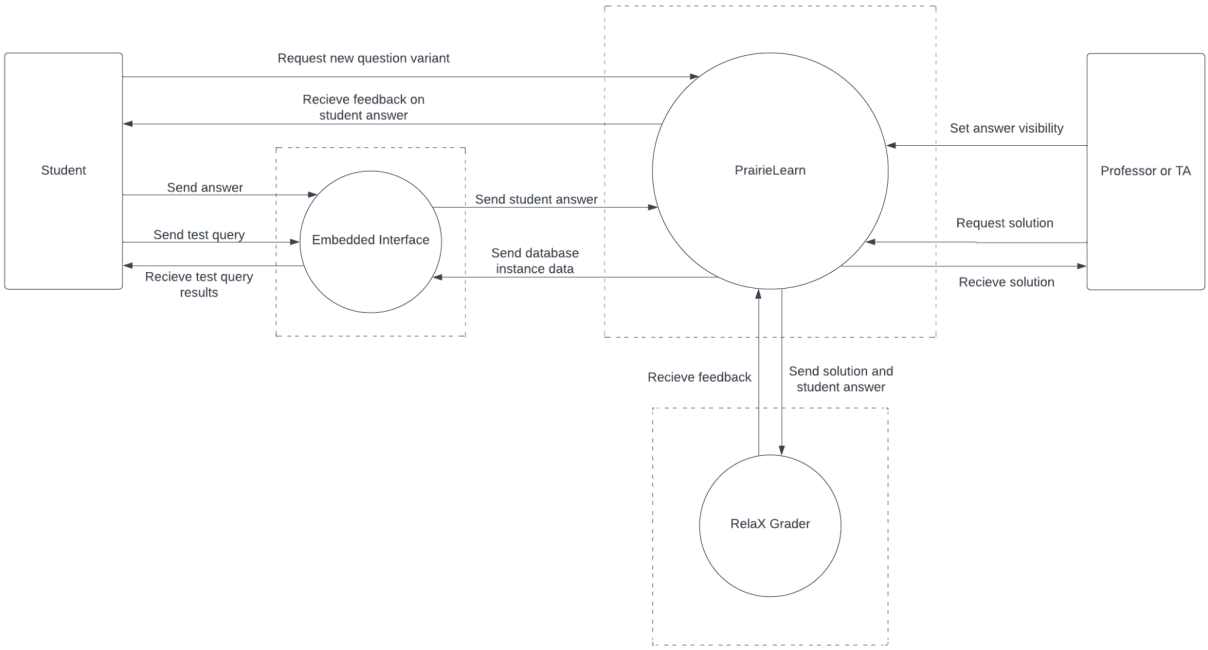


Figure 4.3.1: *The embedded interface is responsible for obtaining the student's query in order to execute it and return visual feedback and to submit it to PrairieLearn for grading. PrairieLearn uses the RelaX Grader to grade relational algebra questions; PrairieLearn will return feedback to the student after their submission regardless of whether it used the RelaX Grader or not. PrairieLearn also responds to requests to create question variants. Finally, PrairieLearn will allow professors or TAs to set question visibility as well as respond to a professor or TAs request to view a question's solution.*

4.3.2. Level 1 Data Flow Diagram

The following diagram (figure 4.3.2) shows how the information given by external actors is transformed by the system. Unlike the level 0 data flow diagram, the level 1 data flow diagram shows how data is transformed within each system.

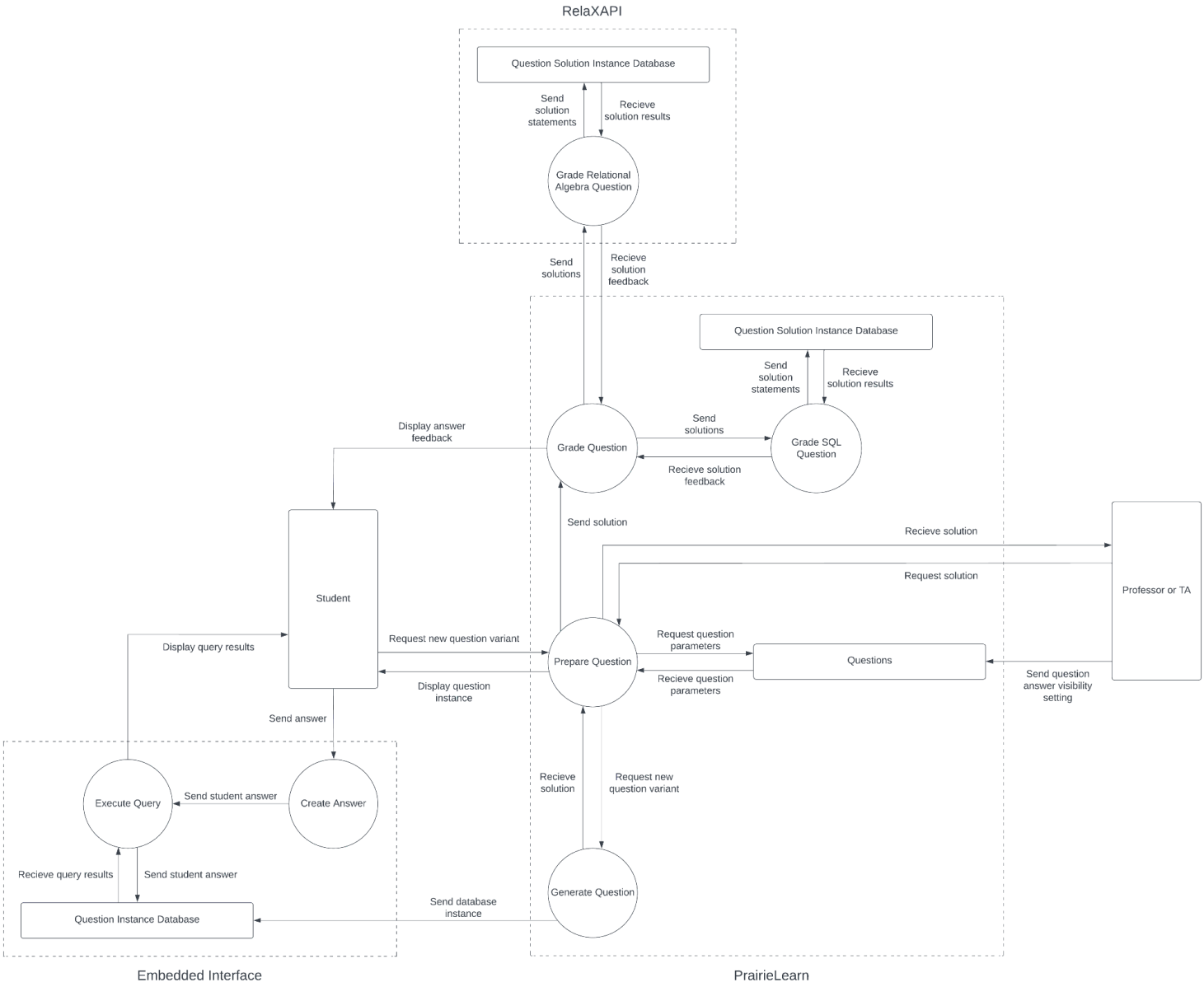


Figure 4.3.2 (on the page above): *This figure has the same information as figure 4.2.1 but has expanded upon the various systems. The embedded interface uses the Create Answer process to obtain the student's answer so that the Execute Query process can run thq query on the question instance database and display its results to the user. PrairieLearn uses the Questions database to obtain question parameters in order to generate a new question variant. This question variant's information is used to instantiate both the Embedded Interface's Question Instance Database as well as either the SQL Solution Instance Database within PrairieLearn or the external RelaxAPI's Solution Instance Database.*

5. UI Mockups

5.1. RelaX Integration

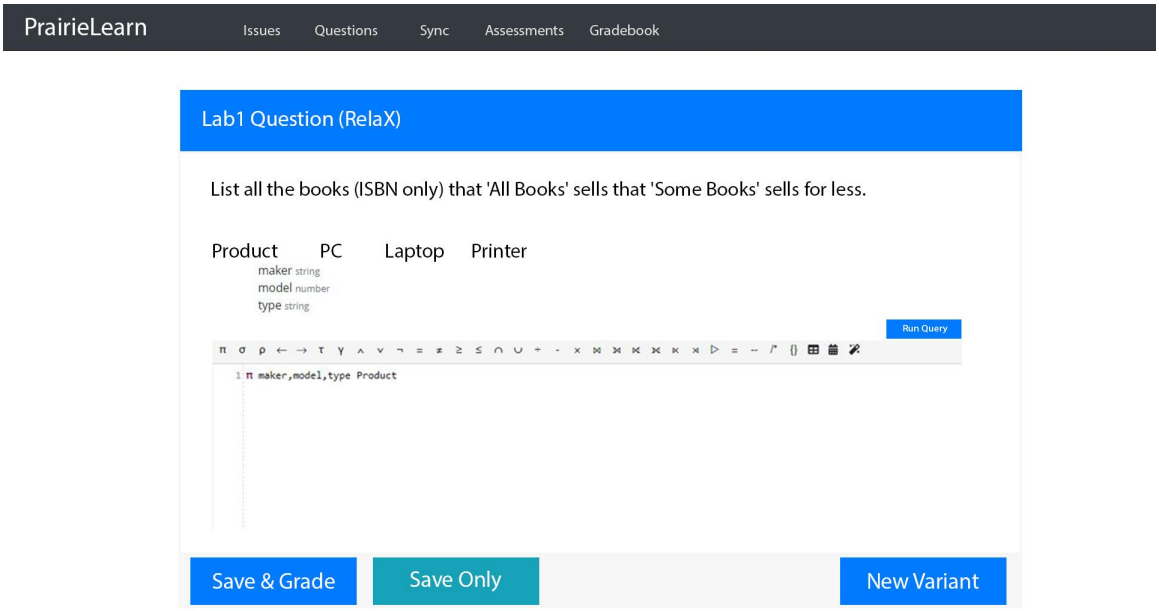


Figure 5.1.1: *A mock UI for the RelaX editor within PrairieLearn.*

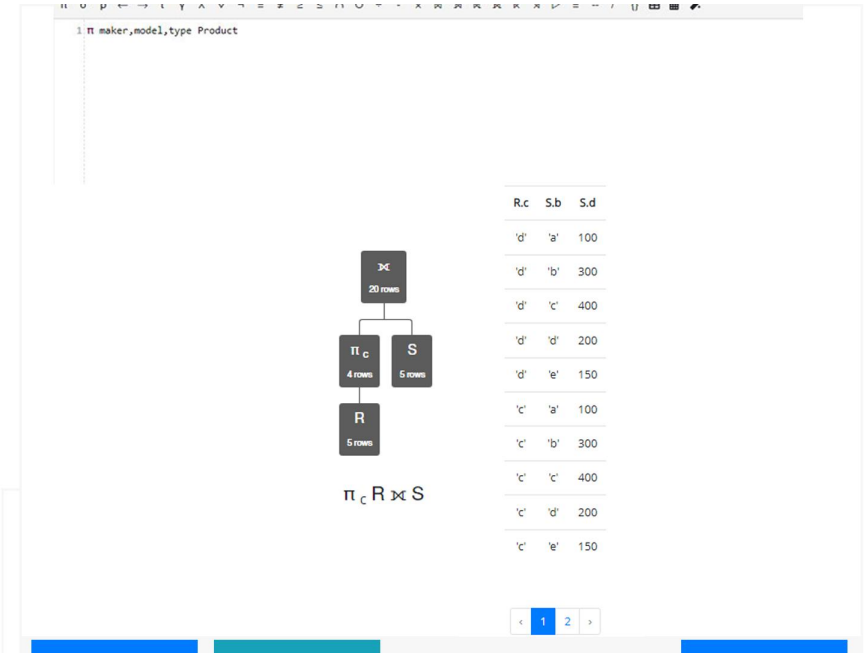


Figure 5.1.2: *A mock UI for the RelaX query results within PrairieLearn.*

5.2. SQL/DDL Integration

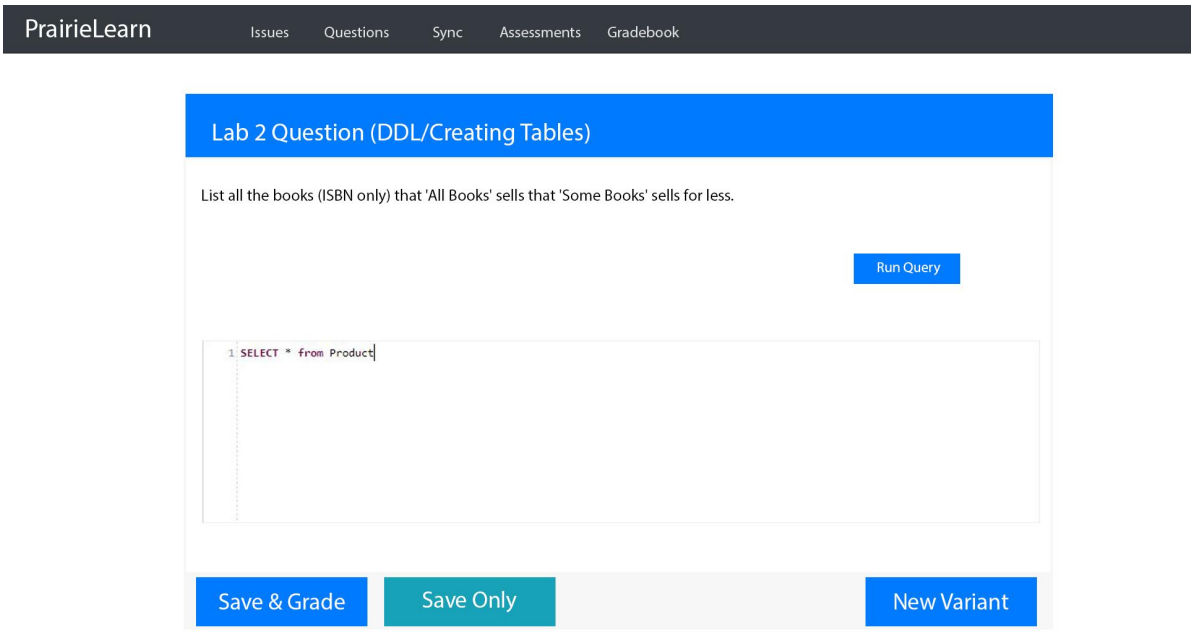


Figure 5.2.1: *A mock UI for the SQL editor within PrairieLearn.*

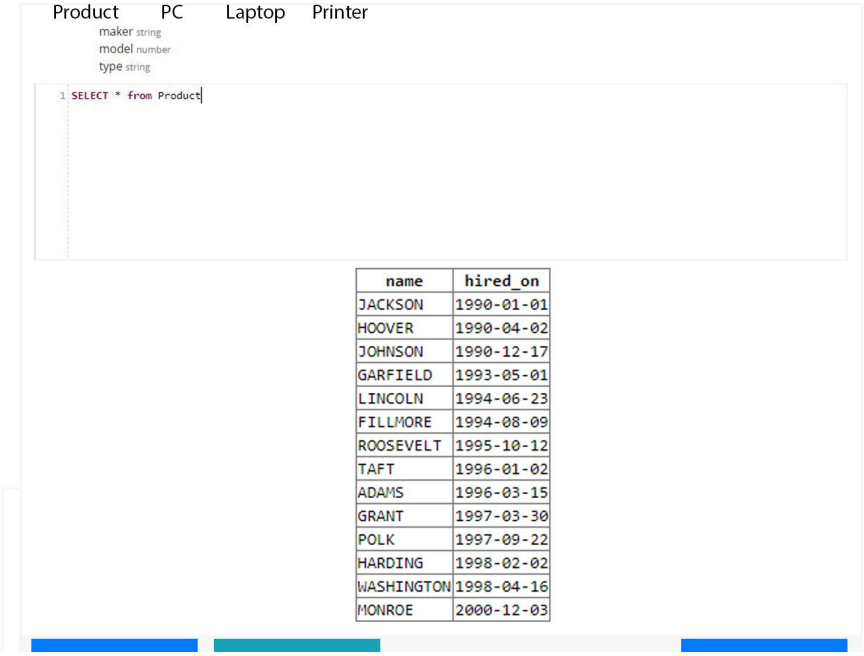


Figure 5.2.2: *A mock UI for the SQL query results within PrairieLearn.*

6. Technical Specifications

6.1. Frontend

Integration and development of front-end development will continue to be in JavaScript, HTML, Mustache, and CSS.

6.2. Backend

Back-end development will continue to be in Python. PrairieLearn is built with Node.js but will not be directly used in our system's development.

6.3. Database

The data for the labs (questions and solutions) will be stored using PostgreSQL. This is also handled through the PrairieLearn system and will not be directly used in this system's development.

6.4. Tools to Build Software

- IDE: Visual Studio Code
- Time Tracking: Clockify
- Task Management: GitHub Projects

- CI/CD: DroneCI

7. Testing

7.1. Overview

Testing is a crucial component of the software development process. It ensures that the system meets all of our specified requirements and functions as the client intends. In this design document, we explain the various types of testing that will be applied and the purpose that it serves. These include: unit testing, integration testing, performance testing, and functionality testing. As we are following a test-driven development structure, these tests will guide our development and reduce the amount of resources dedicated to bug fixes later in development.

7.2. Unit Testing (Structural)

Unit testing is a white-box testing technique. It involves designing tests for each function to ensure that it produces the expected output. At minimum one unit test will be designed prior to a function being created with additional unit tests designed during development. The goal is to achieve coverage testing to a degree to ensure that the majority of our code is tested. However, irrelevant tests will not be designed solely for the purpose of increasing our coverage. An example of a unit test would be an empty submission correctly being given a score of 0 ensuring that the scoring function works even without input. These tests will be written with Python Unittest Framework inside python files and be run with DroneCI.

7.3. Integration Testing

Integration testing is a black-box testing technique. This verifies whether multiple components being used together are working appropriately and producing the correct result. It examines the interaction between subsystems and identifies any issues that may arise from these interactions. Integration tests will be written to guide how features will be designed as a whole. An example of an integration test is appropriately handling user submitted strings in a textbox and ensuring that SQL code is then querying the database appropriately. These tests will be written with Python Unittest Framework inside python files and be run with DroneCI.

7.4. Performance Testing

Performance testing is essential to assess the system's ability to scale appropriately. One of the requirements of the system is to function concurrently between hundreds of students and a multitude of classes. As a result, the system will need to be stress tested to determine whether performance dips below acceptable values. We will progressively increase the submission load on the system and measure the response time. In doing so, we may be able to determine bottlenecks and optimize resource allocation. This will be tested on UBC-O's proper PrairieLearn server by logging the time it takes for questions to load, and for submitted answers

to be autograded. We will simulate this under load to mimic our non-functional requirement of approximately 200 concurrent students.

7.5. UI Testing

UI testing allows us to ensure that various components of the software's UI, both new and old, are working as required. In this project some of the new UI components will be the button for running queries without submitting, displaying results of those queries in tables, an input field for Relational Algebra, and an input field for SQL/DDL. This will be tested with Selenium.

7.6. Software/Technologies

7.6.1. Python unittest

Python code for: the rendering of questions, automatic marking, and automatic question generation will be tested with python unittest framework. We will be using this for our Unit Tests, Integration Tests, and to automate our UI tests.

7.6.2. Selenium

To ensure that our UI components, both old and new are working as required, we will be using the Selenium WebDriver for our UI tests.

7.6.3. Drone CI

Our continuous integration will be completed through Drone CI. All tests will be run and passed before merging and again when integrating new features to the main branch

7.7. Continuous Integration/Deployment

Our continuous integration will be completed through DroneCI. All tests will be run and passed before merging and again when integrating new features to the main branch

REQUIREMENTS		**Type of Test**	**Pass or Fail**	**Contributor**
Functional				N=Nishant, A=Andrei, S=Skyler, M=Matthew
1.1	System will allow for relational algebra statements to be entered.	UI Testing Integration Testing	Pass Pass	N,N

1.2	System will show visualizations of the resulting entered statement prior to submission.	UI Testing Integration Testing	Pass Pass	N, N
1.3	System will automatically mark the relational algebra questions once submitted.	Unit Testing	Pass	A
1.4	System will allow for DDL/SQL code to be entered. System will show resulting tables of queries prior to submission.	UI Testing Integration Testing	Pass Pass	N, N
1.5	System will generate random relational algebra questions. (stretch goal)	Unit Testing Integration Testing	Pass Pass	M, M, S
1.6	System will generate random SQL/DDL questions. (stretch goal)	Unit Testing Integration Testing	Pass Pass	S, S
1.7	System will automatically mark the DDL/SQL questions once submitted.	Unit Testing	Pass	N, N
1.8	Student will be able to see the correct answer if the professor has allowed for the correct answer to be displayed after the question is submitted.	Unit Testing Integration Testing UI Testing	Pass Pass Pass	A, M, N
1.9	Professor will be able to set whether the correct answer will be displayed after the question is submitted.	Unit Testing UI Testing	**	**
1.10	Professor will be able to see the correct answer.	UI Testing	**	**
Non-Functional				
2.1	The system will support all COSC 304 users simultaneously – about 200 students.	Performance Testing	Pass	N, A
2.2	The system will ensure data integrity and preservation so that no data is lost upon submission.	Performance Testing	Pass	N
2.3	The system will display entered queries within 3 seconds at scale and under optimal conditions.	Performance Testing	Pass	N
2.4	The system will return automarked submissions within 5 seconds at scale and under optimal conditions.	Performance Testing	Pass	N, A
2.5	The user interface will match existing software used for COSC 304.	UI Testing	Pass	A, M

Technical Requirements				
3.1	Rebuild RelaX editor and calculator into PrairieLearn	UI Testing Integration Testing	Pass Pass	N, N
3.2	Frontend: JavaScript, HTML, CSS	UI Testing	Pass	N
3.3	Backend: Python, Node.JS	Unit Testing	Pass	N, A, S

***Note**: Tests for displaying the correct answer were never written due to it being an existing tested feature of PrairieLearn.*

Section 3: User Testing Report

1. Overview

This document will go through the results we obtained from our user testing session (Testorama) from August 4th,2023.

2. Grading

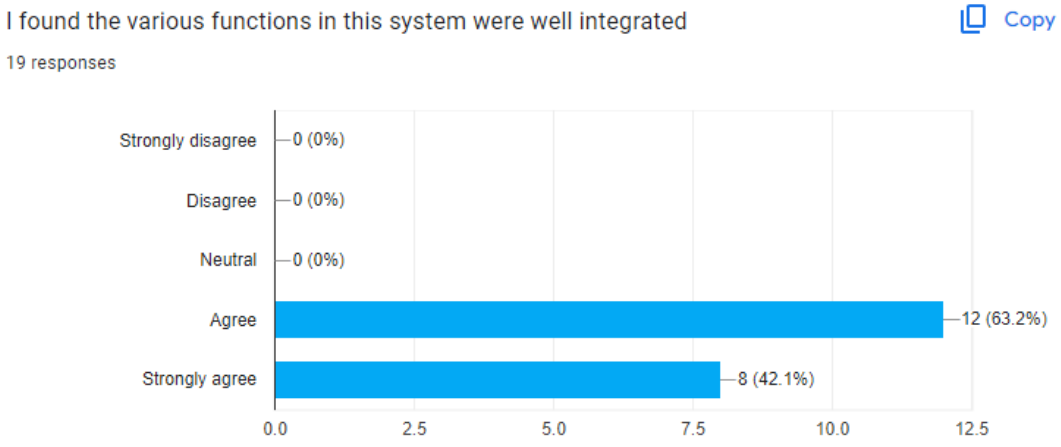
Grading is on a 5-point scale:

1. Strongly Disagree
2. Disagree
3. Neutral
4. Agree
5. Strongly Agree

3. Results

Features:

Users thought that our elements were all integrated very well and we were happy to see them recognize this as the bulk of our project is integration into an existing system.

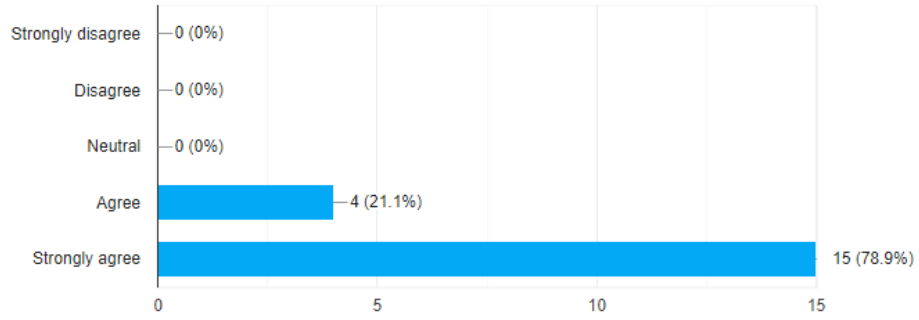


Executing before submitting was a feature our client asked for and the testing users found it to be very straightforward and easy to use the way we implemented it.

I think that the execute before submitting is easy to do

Copy

19 responses

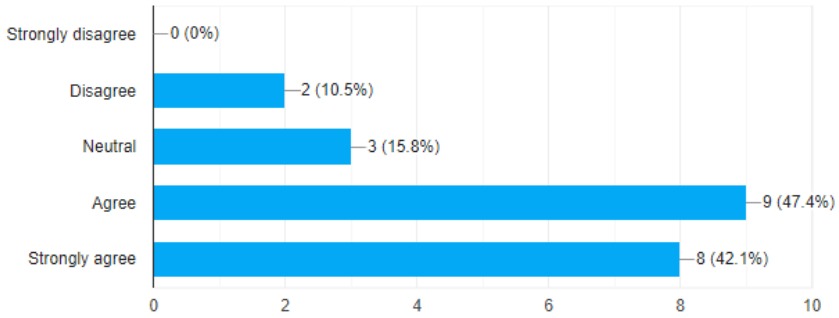


We noticed that while the majority of the students found the autograder to be fair, some felt that it could use improvements. And these improvements have been made to the autograder - we use better weighting of factors and better case-handling. However, it should be noted that “fairness” is very subjective in this sense, hence there will always be some outliers to account for when asking a question like this.

I think that the autograder was fair in assessing my responses

Copy

19 responses

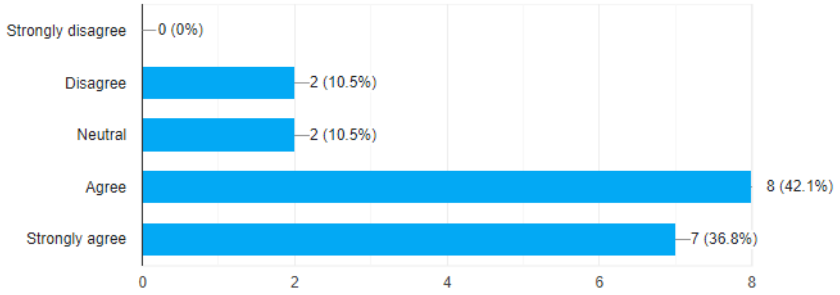


We got similar results with the quality of questions produced by the autogenerator and have made improvements to this feature as well. This is also a subjective question.

I think that the auto generated questions were fair in meeting the appropriate difficulty

Copy

19 responses

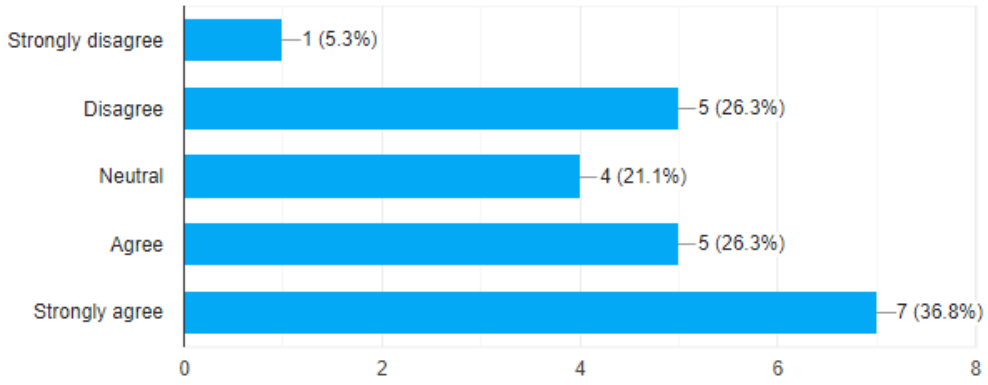


This feature was where we got a lot of feedback from the testing users and this graph is a reflection of that. A lot of people found that it was very confusing to use and needed more context from our team members before they were able to see it's value. This feedback has more to do with how easy it is to figure out how to use and less to do with how easy it is to use in and of itself. These problems have also been resolved.

I think that the schema bar with the dropdown tables is easy to use and contributes to the user experience

Copy

19 responses

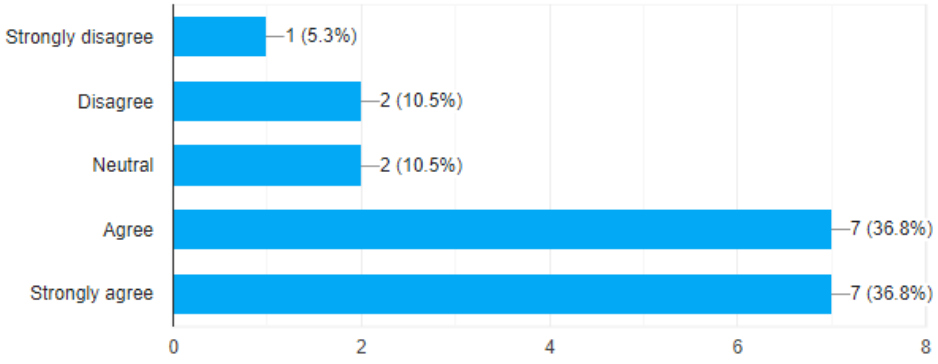


Other than these questions, we found that the results for all SUS questions were positive with an occasional outlier here and there. And from what we gathered by talking to the testing users, their primary issues were with the performance. This is something we also noticed in the weeks leading up to the event - mainly with anything that interacts with databases: autograding and expect output tables

I think that the preview tables for the auto generated QUERY questions load fairly quickly

Copy

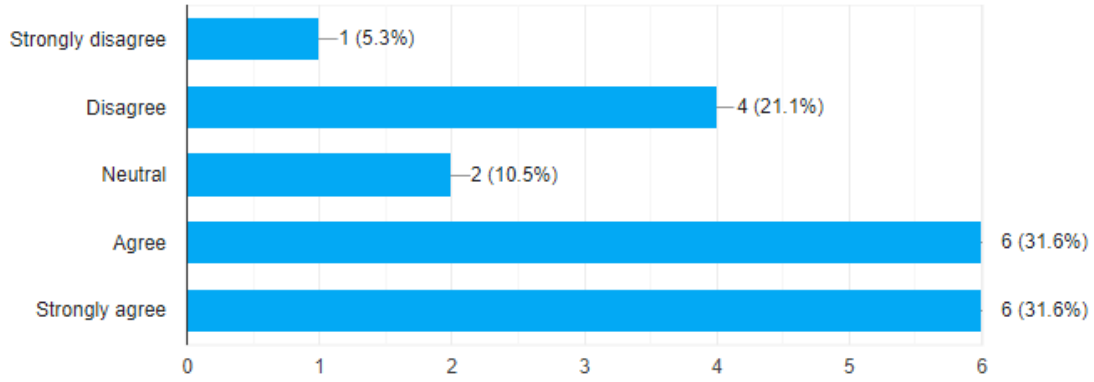
19 responses



I think that the autograder was fairly quick in assessing my responses

Copy

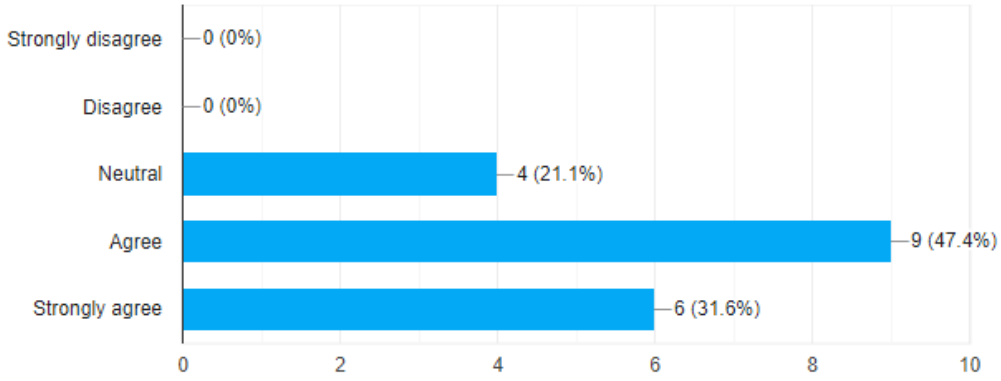
19 responses



I think that the execute before submitting button returns an output fairly quickly

Copy

19 responses



We believe that there are many things affecting the performance of these, and for the most part there isn't much we can do to improve the performance other than using more powerful systems. We did make a small tweak to the autograding which interacts with an SQLite database on the backend to allow for faster interactions. However, in the end there is not much we can do to improve the performance of these from the code itself.

4. Summary

Overall our testing event was very insightful. We got a lot of useful constructive feedback that we were able to quickly address. We also got a lot of encouraging feedback on the progress we've made which is always nice to hear. Our client has been testing the software nearly every week and so we've been able to get a lot of feedback from him on a weekly basis - allowing us to make fixes throughout our timeline.

Section 4: Performance Testing

1. Framework

Performance frameworks used:

- Locust python library
- har2locust python library

2. Overview

Our requirements state that the software must:

- Support all COSC 304 users simultaneously – about 200 students
- Display entered queries within 3 seconds at scale and under optimal conditions
- Return automarked submissions within 5 seconds at scale and under optimal conditions

In order to test this, we first recorded our network activity for the following cases:

- RelaX Static Loading
- RelaX Grading
- RelaX Autogeneration
- SQL Static Loading
- SQL Grading
- SQL Autogeneration

and downloaded them as HAR files. Then we converted these HAR files into locust test files and ran each test with 25, 50, 100, 150, 200, and 300 users to test the average,best, and worst performances of each of these cases with different loads.

3. Results

3.1. RelaX

3.1.1. RelaX Static Loading

Time (ms)	Worst	Avg	Best
0	0	0	0
25	725	256	181
50	2535	1179	211
100	6920	1621	194
150	16384	13333	654
200	22305	16936	161

300	36521	28570	511

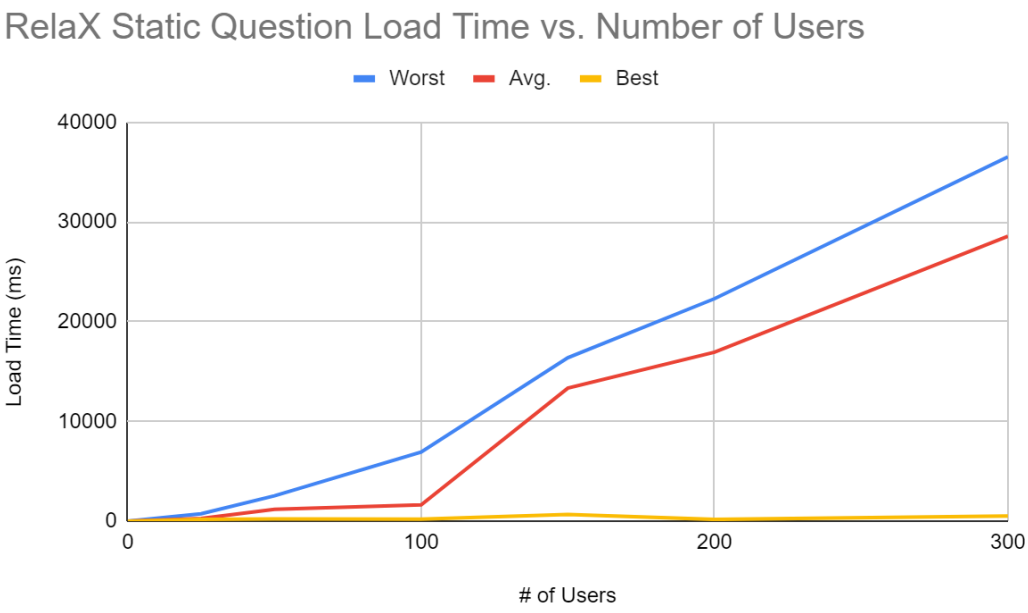


Figure 3.1.1: *The performance for the loading of static RelaX questions increased nearly linearly with the number of users. While the best case remained relatively similar under heavy loads, we can see that the worst and average cases are around the 30 second ballpark in terms of load time, if not more.*

3.1.2. RelaX Grading

Time(ms)	Worst	Avg	Best
0	0	0	0
25	594	205	150
50	2416	1062	168
100	6875	1886	181
150	16499	12256	219
200	22817	15863	359
300	36167	26609	397

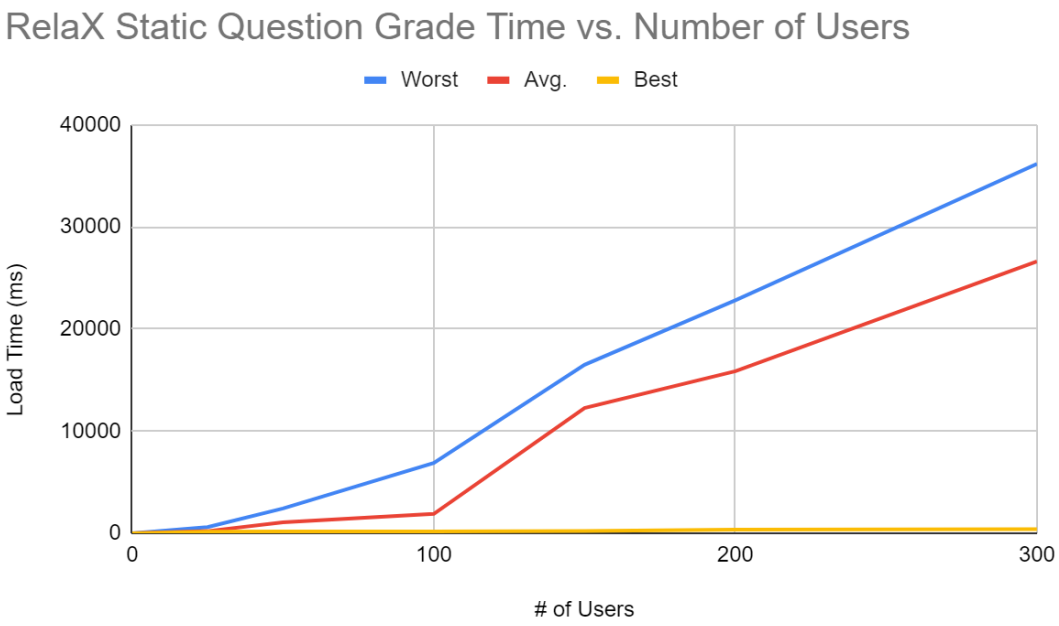


Figure 3.1.2: *The performance for the grading of static RelaX questions increased nearly linearly with the number of users. While the best case remained relatively similar under heavy loads, we can see that the worst and average cases are around the 30 second ballpark in terms of load time, if not more.*

3.1.3. RelaX Autogen Loading

Time (ms)	Worst	Avg	Best
300	37972	19350	225

3.2. SQL

3.2.1. SQL Static Loading

Time(ms)	Worst	Avg	Best
0	0	0	0
25	13405	9461	1781
50	28449	22480	9783
100	49105	24416	15479
150	60887	54129	19667
200	60508	47056	1960
300	60351	47912	271

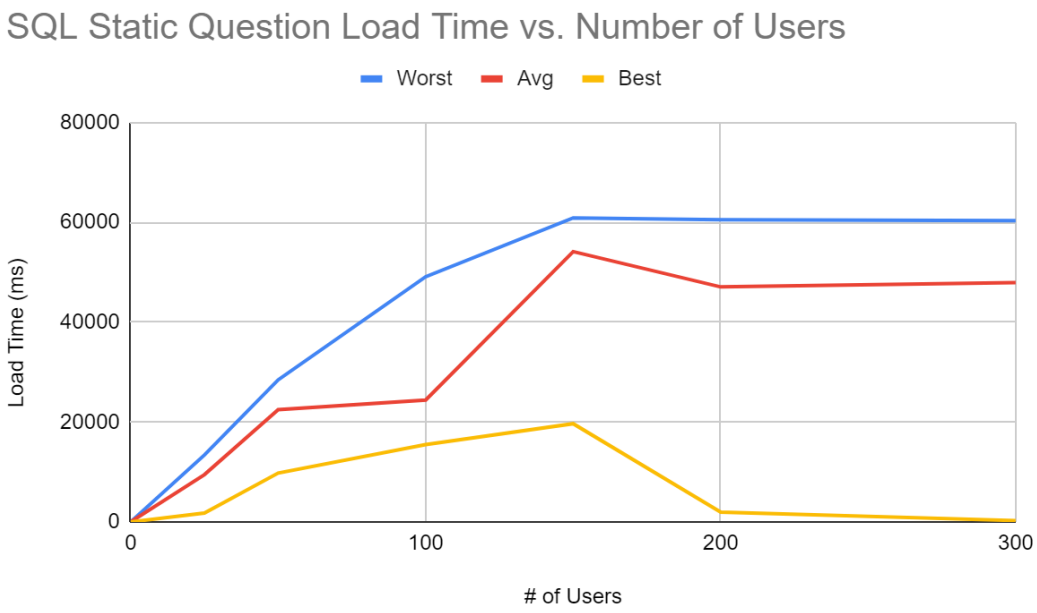


Figure 3.2.1: *The performance for the loading of static SQL questions increased nearly linearly with the number of users at first, but then leveled off after 150. The best case interestingly fluctuates up at a lower user count before declining at a higher user count. We can see that the worst and average cases are around the 45-60 second ballpark in terms of load time at the highest load.*

3.2.2. SQL Grading

Time(ms)	Worst	Avg	Best
0	0	0	0
25	28449	21959	3218
50	68109	50105	8174
100	82539	47148	16460
150	119985	63247	181
200	119971	73138	770
300	106010	60668	334

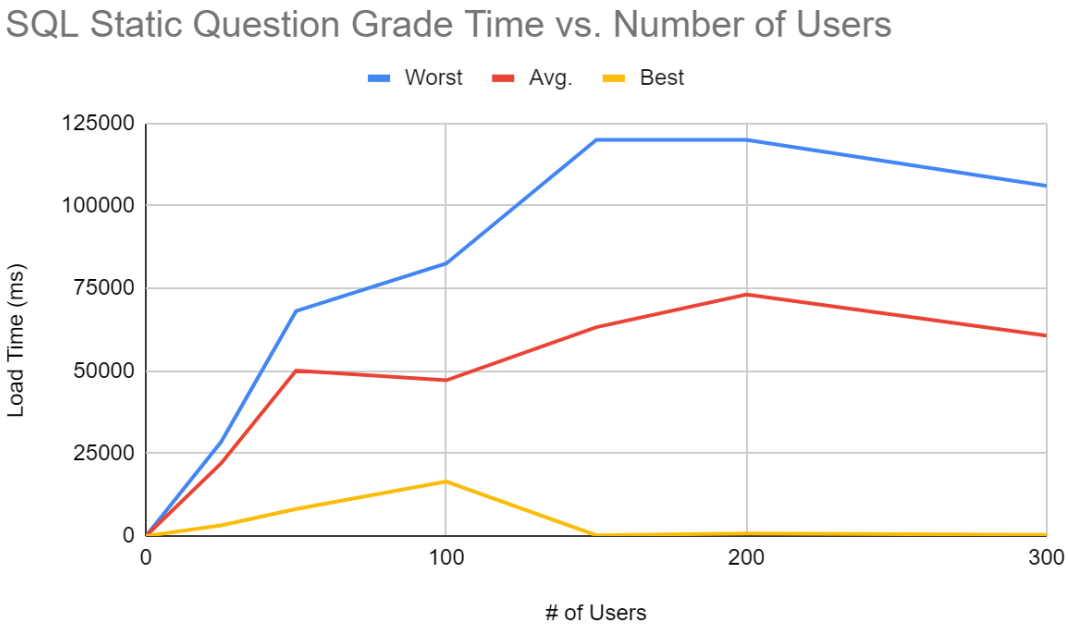


Figure 3.1.3: *The performance for the grading of static SQL questions increased nearly linearly with the number of users at first, but then leveled off after 150. The best case interestingly fluctuates up at a lower user count before declining at a higher user count. We can see that the worst case is between 100 - 125 seconds at high loads, and the average case is about 60 seconds at high loads.*

3.2.3. SQL Autogen Loading

	Worst	Avg	Best
300	26394	16168	236

4. Takeaways

4.1. The performance of the RelaX loading and grading are both very similar and follow a similar growth pattern
4.2. Relax questions are ~3x faster than the SQL in both areas
4.3. SQL Grading follows the same growth pattern as SQL loading but takes more time
4.4. Autogenerated questions take ½ the load time as static in both RelaX and SQL
4.5. Autogeneration is faster than static loading under high loads

Section 5: Product Delivery

1. Code Location

Code repository link:
https://github.com/UBCO-COSC-499-Summer-2023/project-3-a-automating-database-question-generation-automating-db-q-gen-marking-team-a

2. RelaxQueryApi Location

RelaXAPI repository link (**required for grading Relational Algebra questions**):
https://github.com/azipis/RelaXQueryAPI

3. Deploy Location

Course repository link: https://github.com/PrairieLearnUBCO/pl-ubco-cosc304

4. Development Installation Dependencies

- Python (version 3.11.3)
- Docker (version 4.21.1)
- Github
- Visual Studio Code (or another IDE)
- Internet browser with development tools (Chrome / Firefox)
- Github Desktop (Optional)
- Docker Desktop (Optional)

5. Development Instructions

Docker Documentation for PrairieLearn

1. Install docker, docker-compose, and docker-desktop(optional)
2. git clone this repository: `https://github.com/UBCO-COSC-499-Summer-2023/project-3-a-automating-database-question-generation-automating-db-q-gen-marking-team-a.git`
3. In your terminal, navigate to cloned directory.
4. In your terminal run `docker-compose up`
5. To check if the docker container is running, run `docker-compose ps`. if the response is empty, run `docker-compose ps -a`, and it will show the status of the closed docker container.

Creating the relaxAPI container

1. Visit repository found at `https://github.com/azipis/RelaXQueryAPI`
2. Follow directions found at that repository.

6. Deployment Installation Dependencies

- Docker (version 4.21.1)
- Github
- Internet browser

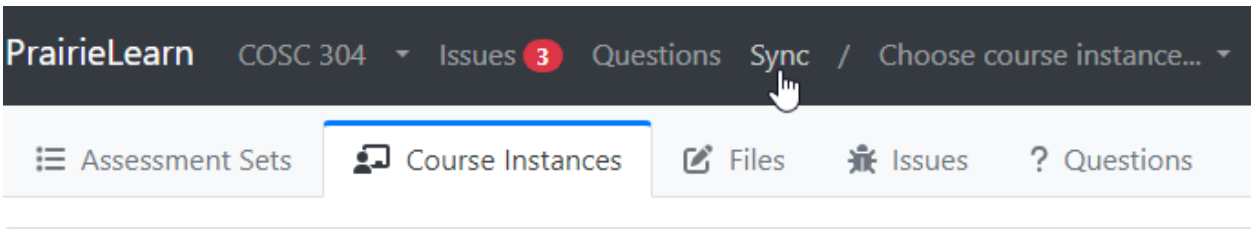
7. Deployment Instructions

7.1. Deploy PrairieLearn using the instructions found here: https://prairielearn.readthedocs.io/en/latest/running-in-production/setup/

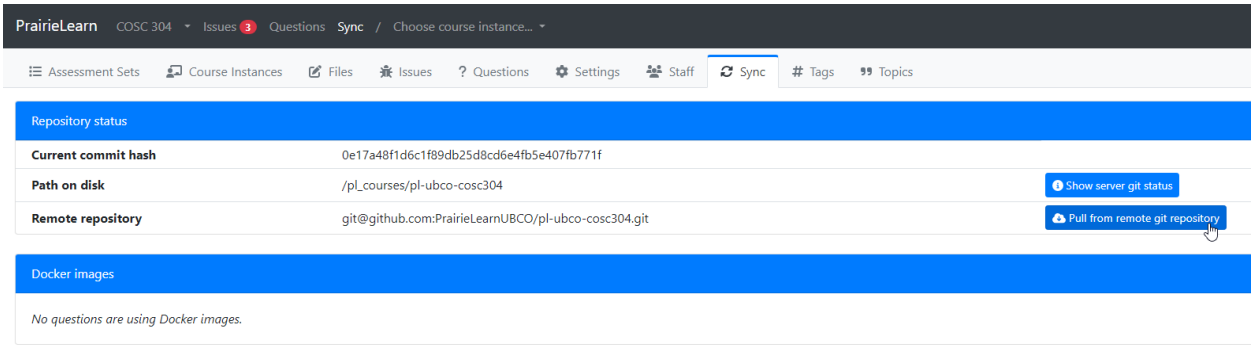
7.2. Push all course changes (elements, questions, additional libraries) to course repository (https://github.com/PrairieLearnUBCO/pl-ubco-cosc304)

7.3. Login to PrairieLearn as an admin/staff user.

7.4. Click sync in the navbar:



7.5. Pull from git course repository:



8. Testing Instructions

8.1. Unit

Install Unittest. There are python unit tests in the sql-element, relax-element, and in the serverFilesCourse/RASQLib folders. Simply navigate to these folders in your terminal and run python -m unittest.

8.2. UI / Integration

Install Selenium. Start the instance on localhost:3000. All the UI/Integration tests are in the tests/ui folder. Once you are there, the UI test files are all in the ddlQuestions, relaxQuestions, and sqlQuestions folders. You need to run python -m unittest in each folder to run all tests.

8.3. Performance

Install Locust. Go to tests/performance. There you have folders RelaX and SQL. In each folder there are 2 subfolders with the test files in them. The performance test files are all named “locustfile.py”. When you’re in a folder, type locust in your terminal and open up the localhost server that the locust interface is on to run your tests with varying loads.

9. Testing Not Implemented

JavaScript unit testing was not implemented for our renderer.js file due to the difficulty of unit testing JavaScript containing jQuery. To clarify, we were not able to figure out a way to mock DOM elements that are required for our nested functions that used a ready/loaded document. With that being said, the functions of these files are being tested through our UI and Integration tests which would fail if our renderer file contained issues.

10. Features Not Implemented

This section is not applicable. All requested features were implemented, working and deployed on live servers.

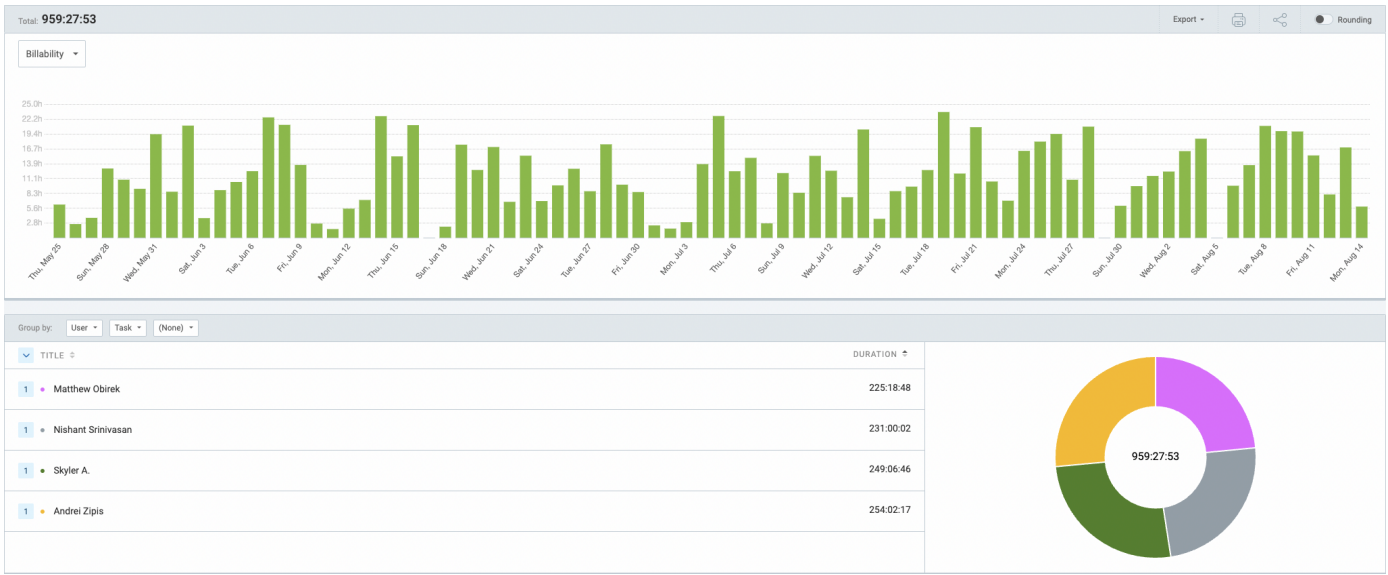
11. Future Work

As our group fully implemented the first three labs of Intro into Databases (*COSC 304*) - Relational Algebra, SQL table creation, and SQL queries - and the previous years group implemented labs 4 and 5 - Database design and UML modeling, and converting UML diagrams to Relational models - Future work with this project could include many different things. We recommend these future projects:

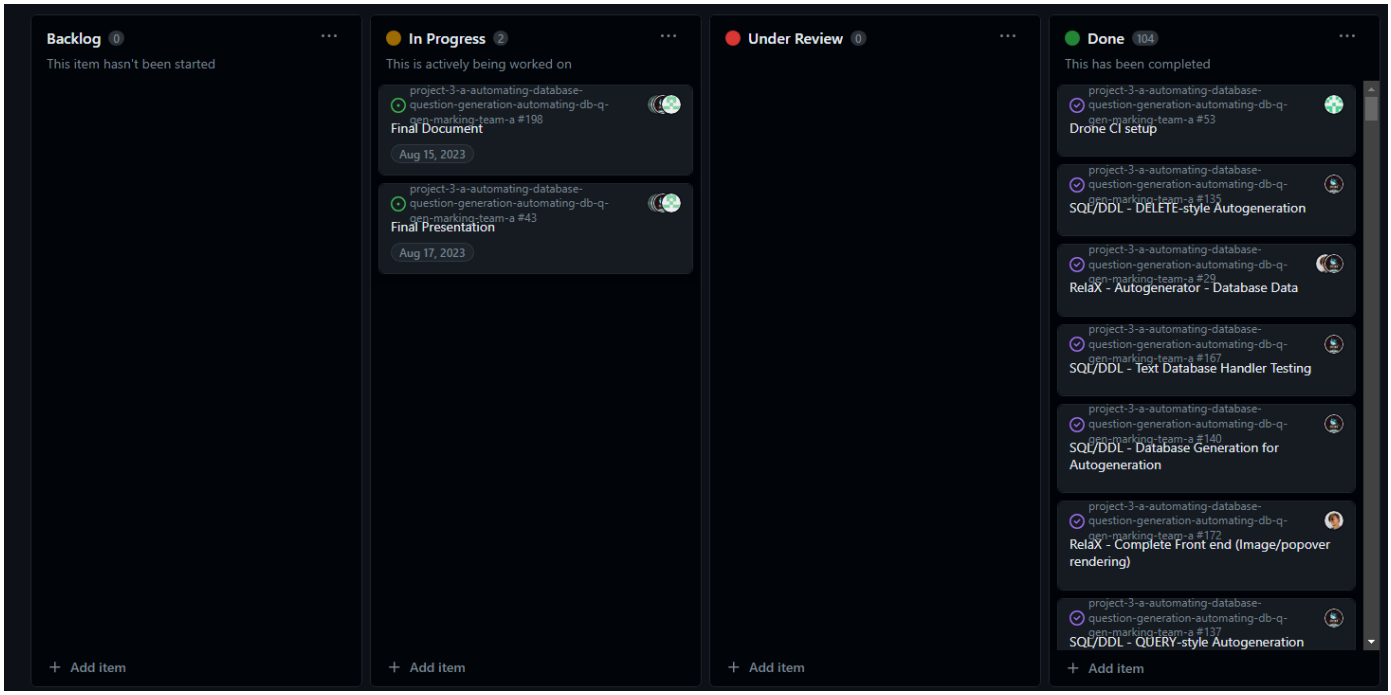
a. Groups could implement future labs on XML, JSON, Views, and triggers. We recommend using the codemirror library, as it is what we used to implement our SQL and RelaX front-end editors.
b. Groups could utilize machine learning, or AI, to improve the grammatical and semantic accuracy of the Automatic generation for the question text and the database layout.
c. Groups could also work on improving performance. We recommend that if the RelaX autograding remains a bottleneck, rewriting the executeRelalg function, and the Relation object, from the Relalg_bundle.js in python using this python library - https://pypi.org/project/dbis-relational-algebra/. The library is built by the same developer of the relax editor. Alternatively, if a faster Relalg_bundle.js is created - one that executes faster - that would solve the performance issue. However, we suggest rewriting it in python because that would reduce the roughly 3 second communication delay between relaxAPI and Prairielearn, and would potentially increase the execution speed of the queries.
d. As we have not extensively tested our deployment with PrairieLearn Team B's gamification scoreboard, we do not know the extent of any conflicts between our projects. Testing, issue resolution, and integration of past projects could be a project in itself, if there happen to be softwear breaking bugs.

12. Workflow Statistics

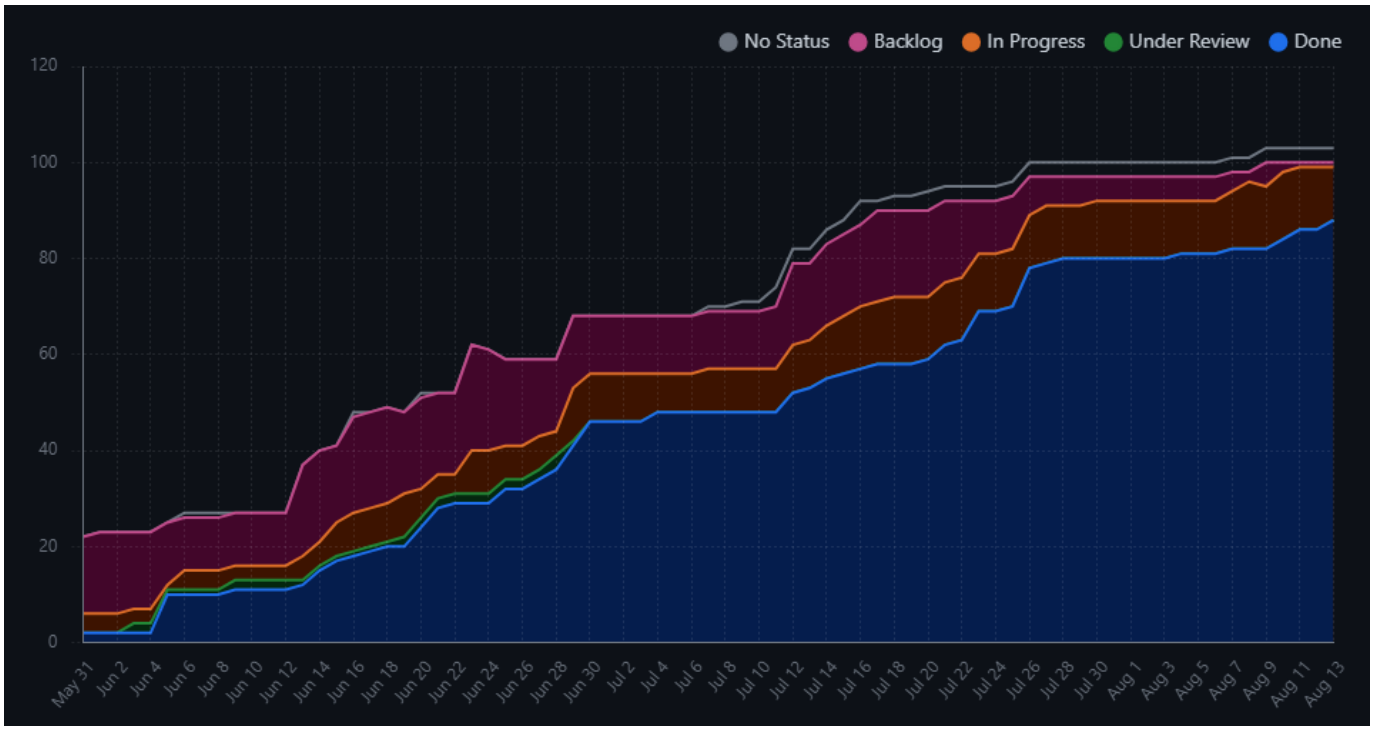
12.1. Clockify Hours



12.2. Kanban Board



12.3. Burnup Chart



Note: Github error in their chart generation. At this point only two issues were still in progress.

13. Known Bugs

- Complex relational algebra generated queries may fail to load or save and grade properly. This is due to a bottleneck in the relalg_bundle.js file and the speed it takes to query complex relational algebra queries. PrairieLearn has a 5 second timeout limit for all the various parts that require communication with the backend. Any requests taking longer than 5 seconds will result in a timeout error.
- SQL question feedback may sometimes display “Code Match” feedback instead of the comprehensive feedback that should be given.

14. Core Requirements Delivered

- Integrate static questions from COSC 304 Labs 1,2,3
- Integrate RelaX Editor
- Integrate SQL Editor
- Execute answers prior to submission
- Display query results and visualizations

- Automate randomized question generation
- Automate grading
- Stretch: Grading for partial marks
- Stretch: Randomized database and data generation
- Stretch: Expected output

15. Lessons Learned

- Not delaying testing until CI/CD setup.
- Always pull before merging and resolve merge conflicts in your own branch.
- Ensure constant communication and feedback with the client. Your approach may not align with their vision of the product.