An Online Tutoring and Assessment System for Teaching Relational Algebra in Database Classes

Hasan M. Jamil(✉), Kallol Naha, and Farjahan R. Shawon

University of Idaho, Moscow, ID 83844, USA
jamil@uidaho.edu, {naha7197,shaw0901}@vandals.uidaho.edu

Abstract. While there are several online tools to practice SQL, except for single tool to practice relational algebra, there practically are no system for teaching, tutoring or assessing relational query language assignments for database classes. In this paper, we introduce *ReliQ*, an online tutoring and assessment system for teaching relational query languages to database students. ReliQ supports a number of features for convenient management of assignments and tests for both practice (in tutoring mode) and authentic assessment (in testing mode). It is capable of grading assignments autonomously and generating useful hints for an effective, enriching and unparalleled eLearning experience.

Keywords: Intelligent tutoring · authentic assessment · eLearning · self-paced learning · user interface · relational algebra · database classes

1 Introduction

As delivery and management of traditional courses increasingly go online, the demands for more sophisticated learning and administrative tools are also growing. Two of the advantages online platforms promise are self-paced learning [24] and scalability [1]. The absence of a physical classroom in online learning also necessitates, among many other support systems, a robust tutoring [9] and assessment [18] system support for effective eLearning. For database teaching in particular, several decoupled tutoring systems were designed to cover topics of a first database class with varying degrees of focus and sophistication. The absence of a comprehensive database teaching, tutoring and assessment system leaves open the opportunity for developing one.

The relational algebra query language tutoring and assessment system, *ReliQ*, we present in this paper is one of the subsystems of a larger and comprehensive database tutoring and assessment system called *Project 360*. Project 360 is designed to include four inter-connected and integrated tutoring systems – Conceptual Database Design (*CoDD*), Visual SQL (*ViSQL* [11]), Relational Algebra Query Language (*ReliQ*), and Normalization and Database Design Theory (*NoDD* [7,8]). In the remainder of this article, we discuss how ReliQ approaches authentic assessment and tutoring of database courses using relational algebra querying.

H. Xie et al. (Eds.): ICWL 2023, LNCS 14409, pp. 62–76, 2023.
https://doi.org/10.1007/978-981-99-8385-8_6

2 Related Research

A recent system for conceptual database design, ERDPlus [10], was introduced. While it allows several sophisticated features to help design and understand ER diagrams and their relationships with SQL schema, it does not particularly support feedback generation or assessment. It is also not part of a tutoring system designed for class management as we discuss in this paper. Similarly, SQL tutoring systems [5,12,14], and normalization tools [22] are not comprehensive systems though they support simple problem solving features toward online tutoring. Of particular interest are the relational algebra tutoring systems RelaX [13] and a cognate system called Alloy [2] for discrete math lab exercises.

RelaX is the closest system that compares well with ReliQ that actually allows execution of relational algebra expressions. It supports most of the relational algebra expressions including several extended operators. Among the contemporary relational algebra tutoring systems, such as radb [23], Relational [21], and IRA [17], RelaX probably is the most advanced and widely used. However, this system is not suitable for a full-fledged tutoring system such as ReliQ. In particular, it does not support assignment posting, tests, or grading. Nor can it be made part of a more elaborate system without extensive redesign even though the source code is made available online. Similar observations apply to radb, Relational and IRA as well.

In contrast, ReliQ is a complete and comprehensive tutoring and assessment system. Instructors are able to design courses around ReliQ, and manage the delivery and conduct of the course in full. It supports a comprehensive set of features traditionally supported by learning management systems (LMS), and more. In particular, relational algebra questions can be asked in assignments and tests, students can be allowed to test their queries before submission, and the tests can be graded fully autonomously. A more sophisticated feedback and query explanation option for ReliQ is being developed along the lines of RATest [15].

3 Relational Algebra Assignments and Tests

A traditional relational algebra assignment usually has a description of a database scheme, a set of table schemes, and a number of queries over the database scheme written in a natural language such as English. A student's task is to implement or translate those natural language queries into relational algebra expressions that when executed will return a response semantically consistent with the queries written in English. For example, the query below

Example 1. List all cats which like to eat the most expensive Purina brand foods.

over the database scheme below where the primary keys are underlined in each table scheme.

Pets(<u>PetID</u>, Name, City, Zip, Age, PetType)
Likes(<u>PetID, Year</u>, FoodID)
Foods(<u>FoodID</u>, Name, Brand, Price)

In ReliQ, in order to compute this query using relational algebra and be graded, a student must write an expression similar (technically speaking, semantically equivalent) to the reference query provided by the instructor. However, for now, let us assume that the student wrote the query below as her answer.

$$R_1\colon\ \Pi_{Name}(\sigma_{PetType='Cat'}(Pets \bowtie Likes \bowtie \Pi_{FoodID,max(Price)\ as\ mPrice}(\sigma_{Brand='Purina'}(Foods))))$$

A more convenient and structured way of composing the query uses extended relational algebra operators such as assignment and renaming as follows.

$$R_2\colon\ Temp \leftarrow \rho_{tmp(FoodID,mPrice)}(\Pi_{FoodID,max(Price)}(\sigma_{Brand='Purina'}(Foods)))$$
$$\Pi_{Name}(\sigma_{PetType='Cat'}(Pets \bowtie Likes \bowtie Temp)$$

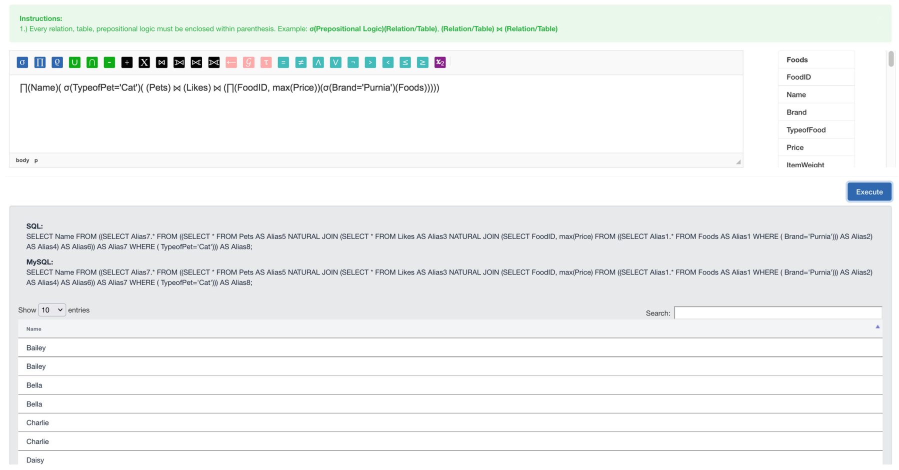
3.1 Question Answering in ReliQ

ReliQ supports a query editor using which users are able to construct relational algebra expressions. Figure 1 shows the ReliQ editor for query construction and execution over a target database. It has three main parts: the operator bank at the top, database table schemes to the right, and the editing pane below the operator bank. Users can choose operators by clicking on the operator icons for ReliQ to insert it in the editor pane at the selected location. Operator arguments or qualifiers (e.g., selection conditions or projection lists) can be inserted as subscripts, or on the same level of the operator, so long the syntax is semantically equivalent, i.e., ReliQ is subscript agnostic.

Fig 1 also shows ReliQ's functional components in an active editing session in which the query R_1 has been constructed and executed against a sample data, e.g., the Pet database. The result of the execution is shown as a table below the editor pane. The displayed table also has column sorting, pagination and page size selection options for convenient visualization. For validation purposes, users are also able to inspect the base table instances just by clicking on the table names at the right segment of the interface. Options to display the standard SQL and MySQL SQL equivalent of the queries in the editor pane are also available as references.

3.2 ReliQ Query Execution

ReliQ query execution is by translation of algebra queries into SQL. In ReliQ, we use MySQL version 8.0.33 on Linux. Since MySQL does not always support or follow standard SQL, ReliQ to SQL translation is MySQL specific, and follows MySQL syntax specific mapping rules. In this section, we present an overview of ReliQ expression to SQL translation procedures, and discuss how student queries are assessed against an instructor provided reference query.


Instructions:
1.) Every relation, table, prepositional logic must be enclosed within parenthesis. Example: **σ(Prepositional Logic)(Relation/Table), (Relation/Table) ⋈ (Relation/Table)**

```
∏(Name)( σ(TypeofPet='Cat')( (Pets) ⋈ (Likes) ⋈ (∏(FoodID, max(Price))(σ(Brand='Purnia')(Foods)))))
```

SQL:
```
SELECT Name FROM ((SELECT Alias7.* FROM ((SELECT * FROM Pets AS Alias5 NATURAL JOIN (SELECT * FROM Likes AS Alias3 NATURAL JOIN (SELECT FoodID, max(Price) FROM ((SELECT Alias1.* FROM Foods AS Alias1 WHERE ( Brand='Purnia'))) AS Alias2)
AS Alias4) AS Alias6)) AS Alias7 WHERE ( TypeofPet='Cat'))) AS Alias8;
```

MySQL:
```
SELECT Name FROM ((SELECT Alias7.* FROM ((SELECT * FROM Pets AS Alias5 NATURAL JOIN (SELECT * FROM Likes AS Alias3 NATURAL JOIN (SELECT FoodID, max(Price) FROM ((SELECT Alias1.* FROM Foods AS Alias1 WHERE ( Brand='Purnia'))) AS Alias2)
AS Alias4) AS Alias6)) AS Alias7 WHERE ( TypeofPet='Cat'))) AS Alias8;
```

Show 10 entries Search:

Name
Bailey
Bailey
Bella
Bella
Charlie
Charlie
Daisy

Fig. 1. ReliQ IDE and editor.

ReliQ to SQL Mapping. Query translation has been leveraged as a less costly implementation strategy in numerous applications [16,20,25] relying on its demonstrated strengths and usefulness [3,6]. The key to assigning translational semantics to a query language is to ensure that the semantic view of the source language is preserved in the target language, and if an inverse mapping to the source language is applied, there will be no "loss". In ReliQ too, we exploit this implementation strategy and translate ReliQ queries into SQL, MySQL version of SQL to be exact, for execution. In the current edition of ReliQ, our focus is on correctness of translation, and a straightforward implementation without a serious focus on efficiency concerns of the back-end query processing costs in MySQL.

Query Classes For the purpose of translation, we recognize that only set operations in ReliQ can be translated directly in SQL. For example, $\Pi_{PetID}(Pets) - \Pi_{PetID}(Likes)$ can implemented as the SQL query

(SELECT PetID FROM *Pets*)
EXCEPT
(SELECT PetID FROM *Likes*);

to discover pets which have no specific food they like to eat. In this query, two subqueries are connected as operands of the EXCEPT operator. In contrast, the query $\Pi_{PetID}(Pets \bowtie Likes)$ cannot be implemented as the SQL query below even though this query too is similar in structure and spirit to the query above.

(SELECT PetID FROM *Pets*)
NATURAL JOIN
(SELECT PetID FROM *Likes*);

Instead, this ReliQ query can be implemented as the following SQL query.

```
SELECT *
FROM (SELECT PetID FROM Pets)
      NATURAL JOIN
     (SELECT PetID
     FROM Likes);
```

Or, more directly as

```
SELECT PetID
FROM Pets NATURAL JOIN Likes;
```

Though both versions appear simple, and the second version simpler yet, the appearances are deceiving. The first version is a direct translation of the ReliQ query that exploits the knowledge that the SQL equivalent of the entire ReliQ query must reside inside a FROM clause, and not much else. In contrast, the second query though more intuitive and cleaner, it requires extensive analysis of the ReliQ query to understand the query sufficiently enough to construct an equivalent SQL version, which in some cases could turn very complicated. We, therefore, chose the first approach for the sake of simplicity even though it is computationally more expensive than the latter.

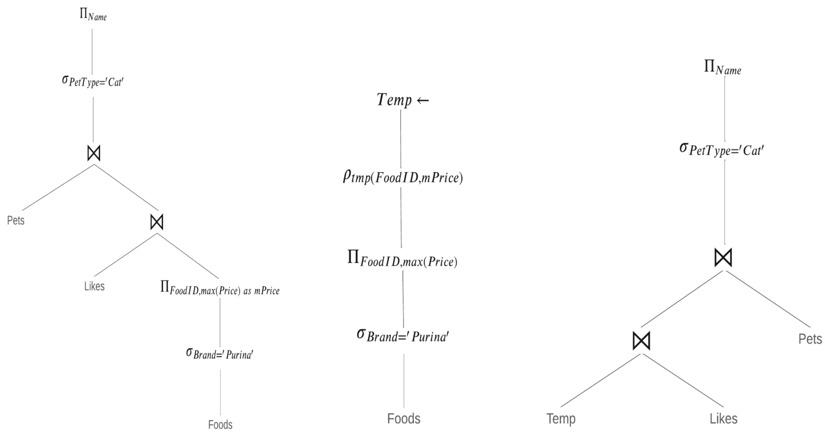
Mapping Algorithms The mapping algorithm we have developed are case by case, extensive and too elaborate to include in this article. Instead, we present the essence of the steps involved in most of these mapping algorithms using the Ex 1 and discuss the general and simple structure they follow on intuitive grounds below.

We first create the query tree from the expressions. Then we proceed to construct SQL expressions step wise from the leaf nodes to the root. The process involves recognizing unary operations (selection, projection, rename, assignment, etc.), and in expression binary operations (joins, Cartesian product, etc.) and open binary operations (union. intersection, difference), and constructing SQL expressions according to the principles discussed above in this section. Special considerations are given to special operations such as division, group by and ordering.

Figures 2(a) and 2(b) show the query trees corresponding to the ReliQ queries R_1 and R_2 respectively in which the leaf nodes are all tables. While they compute the same query and identical responses in two very distinct different ways, we choose to explain the mapping process using the query tree for R_2 in Fig. 2(b) since it covers more operators than R_1. The steps we follow to convert the left query tree of R_2 into an SQL query can be summarized as follows.

Step 1 Identify leaf node *Foods*. Construct expression e as the clause FROM Foods.

Step 2 Recognize unary operation selection. Update expression e as FROM *Foods* WHERE Brand='Purina'.



(a) Query tree for R_1. (b) Query tree for R_2.

Fig. 2. Query trees.

Step 3 Recognize unary projection operation. Update expression e as SELECT FoodID, max(Price) FROM *Foods* WHERE Brand='Purina'.
Step 4 Recognize unary operation renaming. Update e as SELECT * FROM (SELECT FoodID, max(Price) as mPrice FROM *Foods* WHERE Brand='Purina') as tmp.
Step 5 Recognize root assignment node. Create final expression e as

```
CREATE VIEW Temp as
SELECT *
FROM (SELECT FoodID, max(Price) as mPrice
          FROM Foods
          WHERE Brand='Purina') as tmp;
```

Finally, the right query tree in Fig. 2(b) is interpreted as the SQL query

```
SELECT Name
FROM Temp NATURAL JOIN Likes NATURAL JOIN Pets
WHERE PetType='Cat';
```

It does so by recognizing that *Temp* is a virtual table and needs to be computed prior to executing the query above. Also, this time, it starts with the expression FROM *Temp* NATURAL JOIN *Likes* since JOINs are in expression binary operators, and builds up the query above following steps we have outlined above. In contrast, if we were to create an SQL query for the query tree in Fig. 2(a), the query will be as follows.

```
SELECT Name
FROM (SELECT FoodID, max(Price) as mPrice
        FROM Foods
        WHERE Brand='Purina') NATURAL JOIN Likes
            NATURAL JOIN Pets
    WHERE PetType='Cat'
```

Automatic Grading of Student Assignments. While translating student ReliQ expressions into SQL is not too complex a technical challenge, grading them properly somewhat is. Recall that Ex 1 expects the list of cats that eat the most expensive Purina foods and assume that the instructor wanted the name of the cats along with the PetID or Zip to distinguish between the cats with identical names. Then the reference query she wrote is as follows:

$$\begin{aligned}R_r\colon\ &\Pi_{PetID,Name}(\sigma_{PetType='Cat'}(Pets \bowtie Likes \bowtie\\ &\quad(\Pi_{FoodID}(\sigma_{Brand='Purina'}(Foods)) \bowtie\\ &\qquad\Pi_{max(Price)\ as\ Price}(\sigma_{Brand='Purina'}(Foods))))\end{aligned}$$

Incidentally, the reference query R_r is the correct expression for the query in Ex 1, and therefore the student queries R_1 and R_2 are incorrect[1]. The computational challenge here is that how do we test equivalence of the two queries, i.e., $R_r \overset{s}{\equiv} R_1$ or $R_r \overset{s}{\equiv} R_2$ (here, $\overset{s}{\equiv}$ denotes semantic equivalence). Semantic equivalence ($\overset{s}{\equiv}$) technically entails $|R_1| = |R_2| = |R_r|$, and $\forall x(x \in R_1 \Leftrightarrow x \in R_r)$ or $\forall x(x \in R_2 \Leftrightarrow x \in R_r)$.

The queries R_1, and similarly R_2, are not equal to R_r for the obvious reason that they have mismatched schemes. Even if they had matching schemes (just the attribute Name), they are most likely to have different sets of rows, and thus should not be equal. Technically, for the two tables R_1 and R_r, if the conjunct

```
SELECT 'true'
WHERE
            NOT EXISTS          AND    NOT EXISTS
                (SELECT *                  (SELECT *
                FROM R1                    FROM Rr
                EXCEPT                     EXCEPT
                SELECT *                   SELECT *
                FROM Rr)                   FROM R1)
```

holds true, then the two tables are identical in scheme and content. The problem is, in MySQL, EXCEPT requires the order and names of the columns to be identical which is complicated because students potentially could use arbitrary order and apply renaming of the columns. While MINUS does not impose these restrictions, it is not available in MySQL which is what we use as our back-end

[1] It is not just because the student did not include the PetID in the projection, but because the query as written will also include all the Purina brand FoodIDs in the inner projection and thereby making all the non priciest foods also eligible. The reference query R_r first finds the price of the highest priced Purina item, then picks the item's FoodID with an extra join before joining with *Likes* and *Pets* to avoid picking all the lower priced items unlike the student's queries R_1 and R_2.

storage, and thus does not work. While there are other computational[2] and query equivalence theory based [4] solutions, they usually require custom case by case programming, or impose significant restrictions on the class of queries, which is a major hurdle in designing a general good-for-all solution. For example, the ViSQL [11] SQL tutoring system based on Cosette [4] has significant limitations on the class of queries it can support, and was found to be handicapped as a first database class.

To avoid excessive custom coding not guaranteed to work well, we have designed an unorthodox walk around solution to equivalence checking problem that works even though it is slightly inefficient. Given that in a database class, the example databases used are small, and the queries often compute a very small sized table, the solution described below works perfectly well and flawlessly. We first concatenate all the rows in the two tables after converting them into strings, and create two giant strings. Then we sort the characters in the giant strings before generating a hash value using MD5 hash function. The expectation is that if the tables are identical, the hash values will be too. This approach is not impacted by column renaming or order.

4 Implementation of ReliQ

The IDE and the editor in Fig. 1 is the front-end of the ReliQ relational algebra query processor. Though we have abused the usage of the term ReliQ tutoring and assessment system, ReliQ is fundamentally a query processor. What makes it a tutoring and assessment system is its inclusion in the larger Project 360 engine. Project 360 is implemented using HTML, JavaScript, PHP, and MySQL to create a responsive, user-friendly, and feature-rich web-based platform. It consists of two components: the front end and the back end. The front-end of this platform is designed using the Bootstrap HTML framework, coupled with the versatile jQuery JavaScript framework.

The back-end system, on the other hand, is powered by the PHP Laravel framework, serving as the server-side scripting language. The foundation of this platform is the well trusted and widely used relational database management system MySQL. MySQL offers effective tools for data storage, retrieval, and modification, making it a prudent option for managing challenging database tasks. Also its compatibility with Laravel ensures smooth integration and seamless data interactions between the back-end and the database. We leverage Laravel's built-in authentication system to implement secure user registration, login, and password management.

The relational algebra subsystem ReliQ is implemented as a user friendly editor. The main part of the tool's body comprises a highly customized a WYSIWYG rich text editor, called CKEditor, specifically tailored to support relational algebra operators. The editor allows users to write and compose algebraic expressions directly within the web application. A third panel is dedicated to displaying the execution results as shown in Fig. 1 using PHP on the server-side.

[2] https://tinyurl.com/3vs8xkjr.

4.1 Syntax and Semantics

ReliQ supports all standard and extended set of relational algebra operators, and adopts the operator sets of Silberschatz, Korth and Sudarshan [19]. We have iconized the operators that can be clicked to select and place at the editing pane where the cursor is. All unary operators (e.g., select σ, project Π and rename ρ) uses two sets of parentheses – one set for the descriptors (Boolean condition, attribute list or the scheme) and the other for the operand, e.g., $\sigma_{()}()$. Users are able to fill those appropriately. Relation and attribute names can be selected by clicking on the table name or attribute names in the right, The subscripting of the descriptors are optional, and can be toggled by clicking on the X_2 icon at the far right end in the icon panel. Logical operators can also be selected or typed. In the current edition, the parentheses as described are mandatory.

The parentheses requirement is also true for binary operators such as union, intersection and join. For example, *Foods* $\cup$ *Likes* is syntactically incorrect. The correct syntax is (*Foods*) $\cup$ (*Likes*). This is because we are applying uniform bracketing rules across an expression. Since a binary operator can also be flanked by a pair of complicated expressions, we require that they be enclosed within a pair of parentheses to avoid confusion. Thus, the same rule applies to base tables as well. We are considering relaxing the parentheses rules in the future edition of ReliQ as it could become unwieldy at times. Yet, we believe that ReliQ syntax is more intuitive, forgiving, and easy to use than RelaX, radb, Relation or IRA. Furthermore, the group by ($\mathcal{G}$), assignment ($\leftarrow$) and order by (τ) have a more user friendly syntax in ReliQ than any of its predecessors.

4.2 ReliQ Query Processor

As discussed in Sect. 3.2, ReliQ query processor is implemented in MySQL using query translation. We did not include all the mapping algorithms from relational algebra queries to SQL in this paper for the sake of brevity. However, implementation of the assignment operator warrants special consideration and a brief discussion.

The MySQL back-end used by ReliQ for database support and SQL engine can only use one instance of the database. That essentially means all the users logging in to use a specific database are using one single instance. For querying purposes it is not a problem most of the time. However, the assignment operator is implemented using the CREATE VIEW statement at the SQL level. Also, CREATE VIEW in MySQL makes the generated view permanent for everyone. This situation has two implications. First, we cannot just drop a generated view for a given user to clean up the cache because another student may have generated an identically named view, which is truly common for assignment statements, e.g., $x \leftarrow exp$. The flip side of this issue is that once a user created a view named, say x, an assignment statement by another user trying to create another view named x will fail.

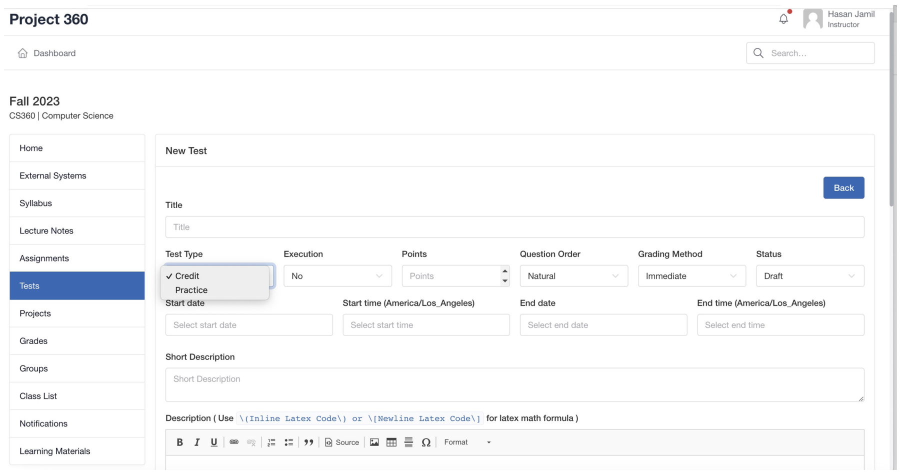
To resolve this issue, we generate a unique ID to the millisecond level and append to each relation name used to the left of the assignment operator. We also create and maintain a list of these tables, and drop the tables at the end of the execution of the query to clean up the cache. This strategy helps to manage the growth of useless tables in the back-end, avoid accidental dropping of active views, and conflicts with views generated by multiple users.

5 ReliQ as an Online Tutoring and Assessment System

ReliQ primarily supports two types of registered users – students and instructors. Students and instructors are allowed access to the public partition of ReliQ where instructors could post assignments, and tests over a test database. All students will have access to these artifacts for self-learning without any assistance from the instructor. However, instructors can open a course, and let students register in their courses in a more closed setting. A course can have defined duration, structure, help and more guided learning. In this section, we discuss the various components of ReliQ that support tutoring and assessment of relational algebra eLearning.

5.1 Instructor's Dashboard

Figure 3 shows the instructor's course dashboard through which she manages the course CS360 she created for her students. Aside from the fact that an instructor is able to do standard functions such as post syllabus, lecture notes, and learning materials, she also can post assignments, and tests, and grades. Finally, she is able to add students to her class. However, for the sake of brevity, we will only focus on how an instructor creates assignments and tests, and grades them.



Project 360

Hasan Jamil
Instructor

Dashboard

Search...

Fall 2023
CS360 | Computer Science

- Home
- External Systems
- Syllabus
- Lecture Notes
- Assignments
- Tests
- Projects
- Grades
- Groups
- Class List
- Notifications
- Learning Materials

New Test

Back

Title: Title

Test Type	Execution	Points	Question Order	Grading Method	Status
✓ Credit / Practice	No	Points	Natural	Immediate	Draft

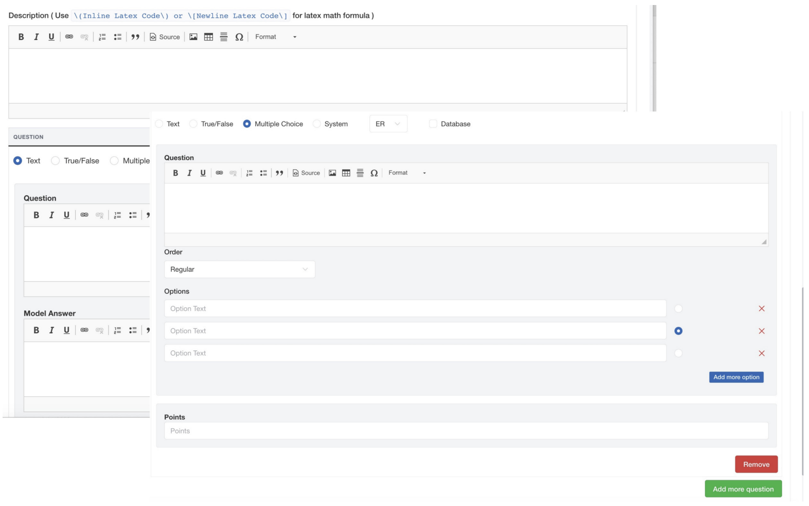
Start date	Start time (America/Los_Angeles)	End date	End time (America/Los_Angeles)
Select start date	Select start time	Select end date	Select end time

Short Description: Short Description

Description (Use `\(Inline Latex Code\) or \[Newline Latex Code\]` for latex math formula)

Fig. 3. Instructor's course dashboard.

Creating Assignments and Tests. One of the main features of ReliQ is designing and administering relational algebra assignments and tests, for both practice and credit. Figure 3 shows the preamble screen using which the instructor designs a test[3]. A test can be designed as a practice test or a for credit test by choosing the option from the drop down list on the left (e.g., *Credit* as shown). Test date and time, total points and the presentation order of questions can be selected. It can be saved as a draft, or published. Two additional features ReliQ supports uniquely are the options to allow query executions on a sample database, and a mode of grading. There are two modes of grading – immediate and deferred. In immediate mode, a grade is returned fully automatically at a preset time, usually after all students have finished taking the test. In deferred mode, however, the response to the tests are collected and saved, and not graded until the instructor manually requests an automated grading.


Description (Use \(Inline Latex Code\) or \[Newline Latex Code\] for latex math formula)

QUESTION

[✓ Text] True/False Multiple

Question

Model Answer

Text True/False [✓ Multiple Choice] System ER Database

Question

Order: Regular

Options

Option Text

Option Text [✓]

Option Text

Points

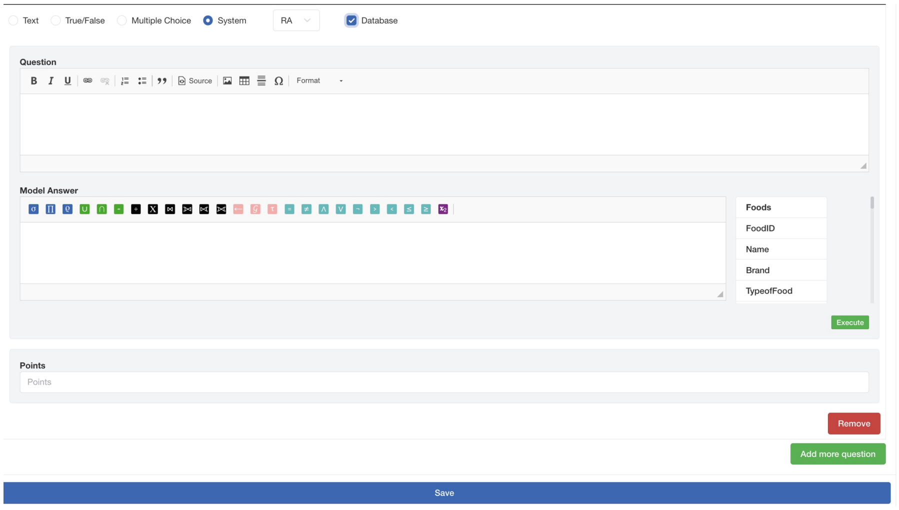
Fig. 4. Assignment creation – text response (inset) and MCQ response (foreground).

Similar to many learning management systems such as Canvas, ReliQ also supports creating questions in an interactive manner as shown in Fig. 4. As shown in Fig. 4, ReliQ supports plain text, "True/False" (not shown in Fig. 4) and MCQ type responses. However, it also supports Code or System type responses that can be directly executed in the back-end database systems. Every question is classified along three axes (if Code type response is selected) – subject category (ER, RA, SQL, FD), database inspection (Yes/No), and execution permission (Yes/No). For subject category "RA", database access selection "Yes" means that for this question, the student will be able to inspect the sample database tables to help her conceptualize query construction. Otherwise, she will have

[3] The process of designing an assignment is almost identical, and hence omitted.

to use her imagination to craft the query. On the other hand, if “Execution” is allowed, the student is allowed to both inspect the database tables and test run her constructed queries on the sample database since permission to execute queries is superseding to database inspection permission. But the converse is not true. Finally, the execution permission at the individual question level is locally overriding permission of the global execution choice the instructor makes in the preamble.

Depending on the response type, the panel for model or reference responses change. Figures 4 and 5 show the model response panels respectively for Text and MCQ, and RA System type responses. The ReliQ IDE shown in Fig. 1 appearing as a panel allows the construction of a relational algebra query, test run it against the sample database, and its correctness confirmed[4]. The model responses are used to auto grade tests and assignments and are never showed to the students unless instructor granted the permission.



Text True/False Multiple Choice [✓ System] RA [✓ Database]

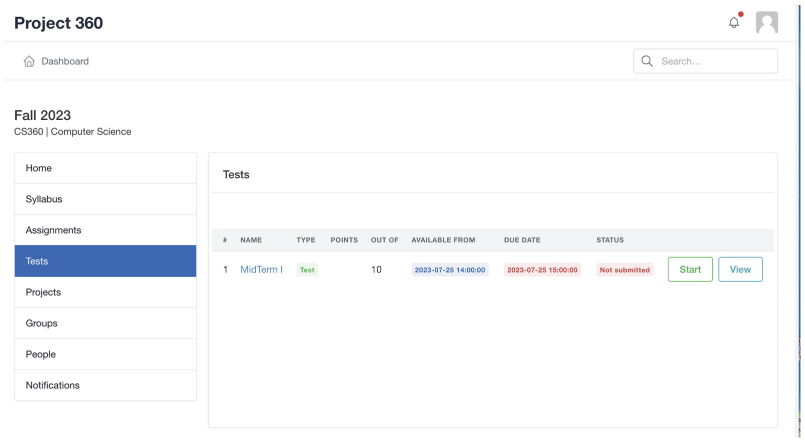
Fig. 5. RA System model response interface.

Automatic Grading. To be able to auto grade an assignment or test, all questions must be of any type except “Text” type responses, requiring manual assessment, in either immediate or deferred grading modes. While the True/False and MCQ type responses are easily graded using standard methods, the RA System questions are evaluated using the approach described in Sec 3.2 against the model response supplied by the instructor.

[4] Note that the ReliQ IDE also serves as a standalone interface for query construction and execution.

5.2 Student's Dashboard

Student's dashboard allows the students to browse published courses, register for courses, and attend the classes asynchronously using ReliQ (as part of Project 360). Additional functions allowed in the course interface are *Syllabus, Projects, Groups, People* and *Notifications*. Navigating to a registered courses, she can see course interface in Fig. 6 after selecting *Tests* (similarly for *Assignments*). The list displays brief descriptions of the test, and various other information related to the test, including test type and time frame when to take it. If already taken, it shows the points earned. It has two buttons – "Start" and "View". A test obviously cannot be viewed before the start of the test, but any time after the start time.



#	NAME	TYPE	POINTS	OUT OF	AVAILABLE FROM	DUE DATE	STATUS		
1	MidTerm I	Test		10	2023-07-25 14:00:00	2023-07-25 15:00:00	Not submitted	Start	View

Fig. 6. Student's course dashboard.

The view of the tests (or assignments) students have is similar to the views of the Figs. 4 and 5. In such test interfaces, they are able to respond to test questions in ways similar to Canvas for Text, True/False and MCQ type questions. But distinctively, in Project 360, they are also able to write executable relational algebra queries directly in the response box for "System" or "Code" type questions, test run them, and save if allowed by the instructor.

6 Conclusion and Future Research

The main focus of this presentation was to highlight the design, functions and features of ReliQ tutoring and assessment system. We have largely emphasized how ReliQ supports construction and execution of relational algebra queries and tested for correctness. As an execution engine, it is similar to RelaX, but its inclusion into Project 360 afforded it more capabilities than any other known contemporary relational algebra tutoring system.

This cloud based ReliQ tutoring system is currently open to instructors for use in their database classes. We intend to collect usage and performance data to share with the community to improve database teaching and learning. We are distinctly aware of the US federal Family Educational Rights and Privacy Act (FERPA) law and keenly compliant with student data privacy. We therefore do not collect student personal identification data, though we collect demographic data to study social and learning related diversity, equity, inclusion, and accessibility (DEIA) issues. We therefore require students personal information to be mapped to non-discernible identities before submitted to the system by the instructors as course participants. Login credentials are then forwarded to the instructors for onward transmission to the students to gain access.

We have plans to enhance ReliQ with semantic feedback capabilities in the direction of RATest [15]. In the current edition, we show the standard SQL and MySQL versions of the relational algebra query. We have plans to also display the constructed query tree for better semantic comprehension of the structure of the queries. Finally, we are considering a feedback system similar to QueryVis [14] and viSQLizer [5] in the near future. These are some of the ideas we plan to seek as our future research.

Acknowledgement. This research was partially supported by an Institutional Development Award (IDeA) from the National Institute of General Medical Sciences of the National Institutes of Health under Grant #P20GM103408.

References

1. Brinkhuis, M.J.S., Savi, A.O., Hofman, A.D., Coomans, F., van der Maas, H.L.J., Maris, G.K.J.: Learning as it happens: a decade of analyzing and shaping a large-scale online learning system. J. Learn. Anal., **5**(2) (2018)
2. Brown, L.E., Feltz, A., Wallace, C.: Lab exercises for a discrete structures course: exploring logic and relational algebra with alloy. In: ITiCSE 2018, Larnaca, Cyprus, July 02–04, 2018, pp. 135–140. ACM (2018)
3. Chan, H.C.: Translational semantics for a conceptual level query language. J. Comput. Sci. Technol. **10**(2), 175–187 (1995)
4. Chu, S., Murphy, B., Roesch, J., Cheung, A., Suciu, D.: Axiomatic foundations and algorithms for deciding semantic equivalences of SQL queries. Proc. VLDB Endow. **11**(11), 1482–1495 (2018)
5. Folland, K.A.T.: viSQLizer: using visualization for learning SQL. In: NIK 2016, Høgskolen i Bergen, Bergen, Norway, November 28–30 (2016)
6. Gansner, E., Horgan, J.R., Kintala, C.M.R., Moore, D.J., Surko, P.: Semantics and correctness of a query language translation. In: DeMillo, R.A., editor, ACM POPL, Albuquerque, New Mexico, USA, January 1982, pp. 289–298 (1982)
7. Jamil, H.: Online tutoring and plagiarism-aware authentic assessment of database design assignments. In: IEEE TALE, Auckland, New Zealand, November 27-December 1 (2023). In press
8. Jamil, H., Shawon, F.: Automatic and authentic eassessment of online database design theory assignments. In: ICWL 2023, Sydney, Australia, November 26–28 (2023). In press

9. Ji, S., Yuan, T.: Conversational intelligent tutoring systems for online learning: what do students and tutors say? In: Kallel, I., Kammoun, H.M., Hsairi, L., editors, IEEE EDUCON 2022, Tunisia, March 28–31, 2022, pp. 292–298 (2022)
10. Jukic, N., Vrbsky, S., Nestorov, S., Sharma, A.: Erdplus. https://erdplus.com/ (2020). Accessed 31 July 2022
11. Karimzadeh, M., Jamil, H.: An intelligent online SQL tutoring system. In: IEEE ICALT 2022, Bucharest, Romania. July 1–4, 2022, pp. 212–213 (2022)
12. Kenny, C., Pahl, C.: Automated tutoring for a database skills training environment. In: Dann, W.P., Naps, T.L., Tymann, P.T., Baldwin, D., editors, ACM SIGCSE 2005, St. Louis, Missouri, USA, February 23–27, 2005, pp. 58–62 (2005)
13. Kessler, J., Tschuggnall, M., Specht, G.: RelaX: a webbased execution and learning tool for relational algebra. In: Grust, T., et al., editors, BTW 2019, 4–8 March 2019, Rostock, Germany, volume P-289 of LNI, pp. 503–506 (2019)
14. Leventidis, A., Zhang, J., Dunne, C., Gatterbauer, W., Jagadish, H.V., Riedewald, M.: QueryVis: logic-based diagrams help users understand complicated SQL queries faster. In: SIGMOD, pp. 2303–2318. ACM (2020)
15. Miao, Z., Roy, S., Yang, J.: Explaining wrong queries using small examples. In: Boncz, P.A., Manegold, S., Ailamaki, A., Deshpande, A., Kraska, T., editors, SIGMOD 2019, The Netherlands, June 30 - July 5, 2019, pp. 503–520 (2019)
16. Michel, F., Faron-Zucker, C., Montagnat, J.: A generic mapping-based query translation from SPARQL to various target database query languages. In: WEBIST 2016, Volume 2, Rome, Italy, April 23–25, 2016, pp. 147–158. SciTePress (2016)
17. Muehe, H.: Interactive relational algebra in latex. https://db.in.tum.de/people/sites/muehe/ira/ (2022). Accessed 22 July 2023
18. Murad, D.F., Wijanarko, B.D., Leandros, R., Murad, S.A.: The effectiveness of online-based authentic learning assessment. In: SIET 2022, Malang, Indonesia, November 22–23, 2022, pp. 277–282. ACM (2022)
19. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts, Seventh Edition. McGraw-Hill Book Company (2020)
20. Song, H., Kim, A., Park, S.: Translation of natural language query into keyword query using a RNN encoder-decoder. In: ACM SIGIR, Shinjuku, Tokyo, Japan, August 7–11, 2017, pp. 965–968. ACM (2017)
21. Tomaselli, S.: Relational - educational tool for relational algebra. https://github.com/ltworf/relational (2022). Accessed 22 July 2023
22. Wang, J., Stantic, B.: Facilitating learning by practice and examples: a tool for learning table normalization. In: BCI 2019, Sofia, Bulgaria, September 26–28, 2019, pp. 35:1–35:4. ACM (2019)
23. Yang, J.: IRA - interactive relationale algebra. https://users.cs.duke.edu/junyang/radb/ (2022). Accessed 22 July 2023
24. Zhang, M., Gu, Z.X., Amer, A., Sidhu, G., Srinivasan, S.: Software suite for self-paced learning. Int. J. Emerg. Technol. Learn. **17**(19), 20–32 (2022)
25. Zhekova, M., Pashev, G., Totkov, G.: An algorithm for translation of a natural language question into SQL query. In: ICERIS, Plovdiv, Bulgaria, October 13–14, 2022, volume 3372 of CEUR Workshop Proceedings, pp. 32–40 (2022)