Computer Assisted Assessment of SQL Query Skills

Stijn Dekeyser **Michael de Raadt** **Tien Yu Lee**

Department of Mathematics & Computing
University of Southern Queensland
Queensland, Australia

{dekeyser, deraadt, leet}@usq.edu.au

Abstract

Structured Query Language (SQL) is the dominant language for querying relational databases today, and is an essential topic in introductory database courses in higher education. Even though the language is syntactically simple, relatively concise, and highly structured, students experience many difficulties while learning to express queries in SQL. In recent years a small number of software tools have been proposed to help students learn to write query statements and to assess their querying skills.

In this paper we compare and evaluate existing tools mainly from the perspective of database theory and practice, but also from a pedagogical perspective. Addressing the deficiencies and opportunities uncovered by the evaluation, we then introduce *SQLify*, a new tool that extends the current state of the art by incorporating semantic feedback, enhanced automatic assessment based on database theory, and peer review to arrive at a richer learning experience for students, as well as consistent assessment results and reduced marking for instructors.

Keywords Computer Assisted Learning and Assessment, SQL, Conjunctive Queries, Query Equivalence.

1 Introduction

Structured Query Language (SQL) is the dominant database language today, comprising commands to define relational schema objects (Data Definition Language DDL) as well as provisions to manipulate data (Data manipulation Language DML). In most introductory level database courses in higher education, learning to write DML query expressions in SQL receives a significant amount of attention. Students are not only taught to write syntactically correct statements, but more importantly to translate a natural language question into a semantically correct SQL expression. This learning process is often difficult.

Researchers [10, 11, 14, 17, 19] have identified several common problems that students encounter while learning SQL. We list some of these:

- It is a burden for students to memorize the database schema, possibly resulting in erroneous solutions due to incorrect table or attribute names. This burden also misleads the students to focus on low-level syntax at the expense of high-level query definition.
- Many students misunderstand the basic elements of SQL and first order logic and the relational data model in general. They have trouble grasping concepts such as joins, universal quantification, grouping and aggregation, and some set operations.
- Students may incorrectly perceive a query problem as being easy [19]. Thus, they need experience thinking about the semantics of questions and expressing them in SQL. Part of the problem is that many common, useful, simple to understand, and potentially easy to express queries are outside the bounds of relational completeness. Conversely, many queries that *can* be expressed with a relationally complete language are difficult to compose or comprehend.
- The declarative nature of SQL is rather difficult for many learners to grasp. It requires them to think sets rather than steps.

To overcome some of these problems, students are typically provided with a relational database management system in which they can experiment and increase their skills. However, this approach only provides students with immediate feedback regarding the syntactical correctness of their expressions[1]. This is insufficient to prepare them for assessment, where course instructors usually place more importance on semantical correctness.

From the perspective of students a simple environment is needed in which they can test query expressions and receive immediate feedback regarding both syntax and semantics.

On the other side of the teaching–learning divide, course instructors often desire a tool that helps them teach querying skills while also enforcing consistency in grading and helping to reduce their marking load, freeing them for more effective teaching tasks. For both parties, a tool that helps raise learning to higher orders of thinking (particularly evaluation) can improve educational outcomes [2].

Because relational query languages are not Turing complete, and because important subsets of these languages allow decidability of query equivalence, tools can be constructed that provide immediate syntactic and semantic feedback. In recent years a small number of tools that offer partial feedback have been

[1] Students may also get some idea of the semantic correctness of their queries from evaluation, but a correct query answer for one specific database instance may be misleading the student to think the query is correct in general. In addition, students don't always know what the correct answer for a given instance should be.

proposed. We evaluate a number of these in the next section.

Motivation. (1) To build on the current state of the art and improve the learning experience for students attempting to master SQL, while helping instructors in assessment. (2) To inform database researchers and teachers of educational SQL tools.

Contribution. This paper contains two distinct contributions. First, we compare and evaluate existing computer assisted SQL tutoring and assessment tools from the perspectives of Databases and Computer Science Education. Secondly, we propose *SQLify* as a tool building upon and enhancing these existing solutions, especially in the areas of improved computer assisted assessment and peer review.

Organization. In Section 2 we review existing tools used for teaching and assessing SQL statements. In Section 3 we then give a detailed description of *SQLify*, before presenting an example run-through of the system in Section 4. Some implementation details, especially concerning automatic query assessment, are given in Section 5. We conclude with a summary and future work in Section 6.

2 Evaluation of Existing Tools

In this section we review a number of tools that are well described in literature.

- *eSQL*, proposed in 1997 in [10] to help teach the concept of query processing. It is not used for query evaluation or assessment.
- *SQL-Tutor*, developed at the University of Canterbury, Christchurch, in 1998 [14]. It provides semantic feedback but is not used in assessment.
- *AsseSQL*, a tool created at the University of Technology in Sydney in 2004 [16]. It provides binary grading of queries submitted by students.
- *SQLator*, a tool created by the University of Queensland also in 2004 [17]. This tool is similar to *AsseSQL*.

A number of other tools and systems exist (e.g. [9] and [11]) but are only partially described in scientific research literature. Interestingly, however, very few papers discuss database teaching in general and most of them are not known to database researchers as they are published in computer science education literature.

We now offer a cross-disciplinary review of these tools, first evaluating the systems from a database theory and practice perspective, and then from a pedagogical perspective. The review is summarized in Figure 1.

2.1 Database Perspective

The tools we studied for this paper were primarily created and described from a computer science education point of view, with minor regard to relational database theory. None of them provide detailed implementation information.

Both *AsseSQL* and *SQLator* [16, 17] use heuristical methods to evaluate whether queries entered by students in a test are correct. This involves running the submitted query on a test database, and comparing the output with that of the query included in the definition of the question. In Relational Algebra terms, the condition that is tested is

$$(a(I) - s(I)) \cup (s(I) - a(I)) = \phi \quad (1)$$

where a is the correct query as supplied by the course team, s is the query submitted by the student, and I is an instance of the database supplied by the course team. This test, however, is only approximate: it is possible for students to cheat by creating simple queries that produce the desired result, especially when they are shown the database instance. Sadiq and others in [17] report that their *SQLator* system appropriately marks a query as correct in 95% of cases when the test queries are relatively easy. This result is sufficient for a beginner's course on SQL, but may be more problematic for more advanced courses. Also, the success of the heuristic depends in part on the database instance used in the test; a badly designed instance reduces the level of correctness of this method.

In [16] Prior and Lister have therefore proposed extending their *AsseSQL* tool to run an additional test on a second database instance not known to the students. While this indeed increases the correctness of evaluation, it is still only a heuristic test.

In database theory it is well known that the class of Conjunctive Queries has the important property that it is decidable whether two queries are equivalent. The CQ class is a significant subset of SQL excluding the set operators and grouping statements. In the introductory *Database Systems* course at the University of Southern Queensland, more than 70% of the time spent on teaching SQL is reserved for such queries. Hence, for this type of query, a computer assisted assessment tool should be able to evaluate correctness of submitted queries with 100% accuracy. For queries that are not in CQ, a heuristic approach can still be used, but any automatic grading tool will need to flag such cases so that the lecturer can intervene appropriately. In Section 3 we detail use of query equivalence decidability results to improve the accuracy of computer-based assessment, and to allow automatic grading of reviews performed by students for their peers where possible.

The existing literature also does not address some practical considerations with regard to database systems. For example, the use of the *distinct* keyword or sorting in a query makes it impractical to test equivalence using only the heuristic described above[2]. Furthermore, both *AsseSQL* and *SQLator* seem vulnerable to SQL injection attacks. These include attempts to make unauthorised modifications to a database by taking advantage of the level of access provided by the interface. Care must be taken to check or rewrite a submitted query before it is evaluated by the database server.

None of the systems we reviewed have support for teaching or assessing Relational Algebra expression writing. We argue that adding such support is very valuable, for two reasons. Firstly, most introductory level relational database textbooks that we are familiar with (e.g. [5, 12, 20]) include the teaching of relational algebra, often *before* teaching SQL. They do so for a variety of reasons but mostly because students who understand the relational algebra are more likely to write better SQL queries: "*relational algebra is the key to understanding the inner workings of a relational* DBMS*, which in turn is essential in designing SQL queries*" [12]. Secondly, the techniques used

[2] CQ equivalence testing is also not sufficient for this purpose.

Feature	*eSQL*	*SQL-Tutor*	*SQLator*	*AsseSQL*	*SQLify*
Modelling of student to individualize instructional sessions	✗	✓	✗	✗	✗
Visualization of database schema	✗	✓	✗	✗	✓
Visualization of query processing	✓	✗	✗	✗	✓
Feedback on query semantics	✗	✓	✗	✗	✓[a]
Automatic assessment (using heuristics)	✗	✗	✓	✓[b]	✓[c]
Automatic assessment (using CQ query equivalence)	✗	✗	✗	✗	✓
Use of peer review for assessment	✗	✗	✗	✗	✓
Relational Algebra expressions support	✗	✗	✗	✗	✓[d]
Special treatment of DISTINCT and ORDER BY	✗	✗	✗	✗	✓
SQL-injection attack countermeasures	✗	✗	✗	✗	✓

Figure 1: Comparison of existing tools and *SQLify* detailed in the remainder of this paper. (a) in practice mode only. (b) on two instances (proposal only). (c) for queries not in CQ. (d) planned for next version.

in automated SQL teaching and assessment tools can be readily used for the relational algebra as well, requiring only a user-friendly (and, desirably, pedagogical) interface for entering relational algebra statements, and additional logic to convert students' algebra expressions into SQL, the latter of which is a well-documented procedure.

2.2 Pedagogical Perspective

eSQL [10] was one of the earliest tools proposed for teaching database concepts. This is a system similar to a normal query interface except that the response to a SELECT statement is not merely to show the result, but also to show a sequence of images giving a step-by-step account of how the query result is determined. Hence, *eSQL* visualizes query processing, at least at the conceptual level. This helps students develop a mental model and enhances students' understanding of the semantics of SQL. One of the steps being visualized is the creation of the cartesian product of the input tables. Since the number of rows in the intermediate result may be far too large to show, the system uses an ingenious algorithm to choose a sample row set for display.

The main contribution of *eSQL* is therefore pedagogical. The tool is not meant to analyze queries submitted by students, nor is it used in assessment.

SQL-Tutor [14] is a knowledge-based system that supports students in learning SQL. It focuses on the individualization of instructional sessions towards a particular student, by developing a model of the student's knowledge, learning abilities and general characteristics and tailoring instructional actions to the student's needs. *SQL-Tutor* is also an Intelligent Teaching System designed as a guided learning environment, which helps students in overcoming the difficulties in learning SQL. Students are given opportunities to discover things by themselves; they can learn by doing.

A major strength of *SQL-Tutor* is that this system gives meaningful feedback on the semantic correctness of queries in addition to feedback concerning syntax. Moreover, there are five levels of feedback in the system, yielding increasingly detailed information.

Another feature of *SQL-Tutor* is the visualization of the database schema. This removes cognitive load for students, allowing them to focus on higher level query definition problems instead of low-level syntax.

However, *SQL-Tutor* does not visualize the way a query is executed as *eSQL* does. In addition, the tool only focusses on helping students practice before assessment and does not help instructors in conducting assessment.

Turning to the systems which support assessment, both *SQLator* and *AsseSQL* [16, 17] apply only *binary* grading to queries submitted by students, and do not provide comments or suggestions for improvement. While Prior and Lister [16] argue the sufficiency of this *right-or-wrong* approach, binary grading does not correct students' misunderstandings or encourage further learning.

As well as giving students query problems to solve, *AsseSQL* also shows the desired result of the query they are to write. This is justified as an attempt to overcome students' poor English skills, but creates an unauthentic setting for student learning, as database programmers typically do not know the result of a query before submitting it to the server.

Both *AsseSQL* and *SQLator* create only a single channel of communication between the student and the

instructor via the system. No other forms of communication (e.g., peer to peer) are mentioned as being part of these systems or used along-side these systems.

None of the tools we examined use *peer review* as part of learning and assessment. According to Saunders [18] peer learning is advantageous as "it offers the opportunity for students to teach and learn from each other, providing a learning experience that is qualitatively different from the usual teacher-student interactions". Peer review can take several forms. One form takes a student's submission and allows it to be reviewed by a number of student-peers, a process overseen by an instructor. Peer review has been successfully incorporated in the assessment of student work in various fields, including computing [6, 7, 13]. Peer review allows students to evaluate the work of others which requires higher order thinking skills [2] through evaluating the work of peers and reflecting on their own work. With peer review, students also receive feedback from more than one source enriching the learning experience for students. Receiving feedback from peers can encourage a community of learning [3] which can in turn further encourage higher order thinking. Peer review involves students in the assessment process, encouraging increased engagement in the course and ultimately improved learning outcomes [6]. Peer review, when used as an assessment tool, can also reduce the assessment workload of instructors.

3 *SQLify*

Having compared and evaluated existing computer assisted learning and assessment tools both from a Computer Science Education and a Database Theory point of view, we now turn to the description of *SQLify* (pronounced as *squalify*) which aims to improve existing solutions on several different fronts.

From the discussion in the previous section, it is clear that combining semantic feedback, an enhanced automatic assessment algorithm, and peer review will produce better outcomes for students and instructors alike. Specifically, the following requirements have driven the design of *SQLify*:

- Provide rich feedback to students in an automated and semi-automated fashion;
- Reduce the need for recall for students by presenting the relevant relational schema;
- Illustrate the execution strategy for queries submitted by students to deepen their understanding of database systems;
- Employ peer-review to enhance learning outcomes for students (through students conducting evaluations and receiving feedback from more sources);
- Use a query equivalence testing algorithm combined with peer review effectively to yield a wider range of final marks;
- Automatically judge the accuracy of reviews performed by students as part of their assignments;
- Reduce the number of necessary moderations conducted by instructors, freeing them for other forms of teaching;
- Increase the consistency of marks allocated to students.

Hence, the main focus of *SQLify* is computer assisted practice and assessment using a sophisticated automatic grading system in combination with peer review.

3.1 Use of *SQLify*

The *SQLify* system is intended to assess a student's query writing skills through an online interface in the context of assignments and preparing for assignments[3]. Student use of the system can be seen to fall into a series of phases.

1. Trial and submission
2. Reviewing peers' submissions
3. Receiving feedback and marks.

Students will submit solutions to a number of problems. The value of their submission will be judged by peers, the *SQLify* system and ultimately by the instructor (see Figure 2).

Students complete reviews of (usually two) other students submissions for which they are awarded marks. The accuracy of their submission determines the mark they receive for reviewing.

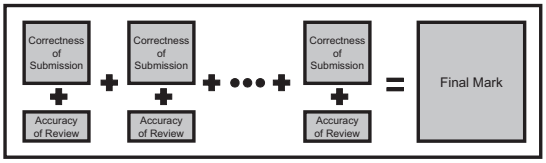

Figure 2: Components of a student's mark.

Finally the marks they received for submission and the accuracy of their reviews is summed to form a final mark.

The following subsections describe in detail the three phases mentioned above.

3.1.1 Trial and Submission

Students are able to develop and trial their query answers to a specific set of problems using *SQLify* and immediately see how the automatic grading system evaluates their work. The *SQLify* system will give one of (a limited set of) the levels of correctness shown in Figure 3. Students may practice query problems indefinitely prior to starting work on assignments. The mark they are shown during this trial period is not necessarily what they would receive from the instructor for the correctness of their submission in assignments; this is given later by the instructor under advisement of the student's peers and the *SQLify* system. When the student is happy with their work they may proceed to submitting query answers to assignment problems.

Students completing assignments using *SQLify* will typically be given a number of English-language problems (say three to five) that he or she would translate to SQL[4]. The problems are well defined descriptions of authentic, real world problems. Students' query answers are submitted through a web form; a screenshot of *SQLify* is shown in Figure 4.

[3]The system is not meant for use in the context of examination.
[4]Or Relational Algebra, as detailed in Section 6.

Level	Description	Students can use	System can use	Instructor can use	Example value
L0	Syntax, output schema, query semantics incorrect	✓	✓	✓	0%
L1	Syntax is correct, schema and semantics incorrect	✓	✓	✓	20%
L2	Syntax and schema correct, semantics are incorrect	✓	✓	✓	30%
L3	Syntax and schema correct, semantics largely incorrect			✓	40%
L4	Syntax and schema correct, semantics seem largely incorrect (not sure)	✓			70%
L5	Syntax and schema correct, semantics just adequate			✓	80%
L6	Syntax and schema correct, semantics seem largely correct (not sure)	✓	✓		90%
L7	Syntax, schema, and semantics are correct	✓	✓	✓	100%

Figure 3: Levels implied by evaluation sentences. Different levels may be used by reviewing students, the *SQLify* system, and by instructors. Internal assessment values (last column) are *example* values for each level; they may be changed by instructors using *SQLify*.

The problems that students are asked to solve relate to differing database schema, so the student is presented with the correct schema (and a sample instance) for each individual problem. The student can also be supplied with hints and comments, and also with the desired output schema for the query (not the desired output instance), if so determined by the creator of the problem.

Once a query is submitted to the system it is checked for SQL injection attacks. First, tables referenced in the FROM clause of the submitted statement will need to appear in the source database schema, or the query will be rejected. Second, the WHERE clause will be analyzed and possibly rewritten using mainstream SQL injection countermeasures.

In *SQLify*'s assessment mode, students will not be notified if their submissions contain queries that are syntactically incorrect (although they should have been able to determine this themselves by trialing their submission in a database). Demonstrating understanding of the particular SQL syntax of the database system that is used by *SQLify* is a part of the assignment. Also, students may use any resources they like while answering the problems, including SQL language reference guides.

Students receive feedback about their submission in the final phase (see Section 3.1.3).

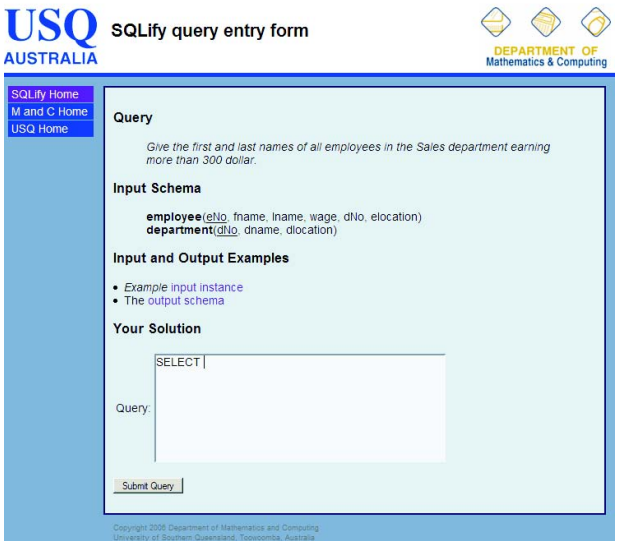

Figure 4: *SQLify* screenshot of the query entry form.

3.1.2 Reviewing Peers' Submissions

After submitting, most students are able to immediately proceed to complete reviews allocated to them. A small pool of early-submitting students (usually four) must wait until enough submissions have accumulated before they can proceed to reviews. Such early-submitters are informed that they must wait and when this minimum pool has been reached the system will automatically allocate reviews to the initial pool and inform them by email that they may start reviews.

This single step submit-review process has been successfully applied [6] and has several advantages over a two step process (submit before deadline, review after first deadline and before a second deadline):

- only one deadline is needed,
- the majority of students are not required to return to the site for the sole purpose of completing reviews,
- students review the task they have just completed,
- students receive feedback from peers shortly after submission, and
- students can work ahead in the course.

The disadvantage of a *single phase* review allocation system is that it must distribute review allocations in a way that maintains anonymity. If students can predict who they will review, collusion between students is possible. This can be countered by complicating the review allocation process and keeping its workings secret, by requiring each submission to be reviewed by more than one peer, and by comparing the accuracy of a student's review to that suggested by the system.

When the system has allocated reviews to a student, reviewing can commence. The student is presented with a similar screen to what they used to input their query answer during the initial submission phase, but where they were previously able to enter their answer the system now shows a read-only query given by a peer. The reviewing student additionally sees the result of applying the query on the relevant database instance. The reviewing student then selects a level described by a sentence from the list shown in Figure 3

that best describes their assessment of the correctness of the query answer. The list of possible levels given in Figure 3 shows all available levels of which the reviewing student may choose levels marked with a tick in the column titled "Students can use". No corresponding internal values are shown to the reviewing student. Reviewing students may express uncertainty by choosing a sentence that includes "not sure". This allows the system to assign a wider range of marks to reviews, but is also used to flag potential problems that need to be moderated by an instructor.

By linking automatic assessment of queries with reviews given by students, it is not only possible to evaluate the correctness of queries, but also the accuracy of reviewers in judging that query. Students will review the work of two peers knowing that the accuracy of reviews they perform will also be assessed.

A student's review accuracy should be marked high when the level they selected for a peer's query answer is very similar to the level ultimately determined for that query answer by the instructor. Conversely, accuracy should be marked low when it differs greatly from the instructor's correctness mark. Hence, the formula for marking accuracy of a review performed by a student is quite simple.

$$\text{accuracyMark} = 100 - |\ \text{correctnessMark} - \text{studentMark}\ |.$$

In other words, the mark given to a reviewer for the accuracy of their review depends on the difference to the correctness mark assigned by instructor. Note that this formula has the additional affect that when a student has signaled uncertainty (by picking level L4 or L6) they will not be awarded full marks for this review.

Giving fellow students a false high or low level evaluation which differs for the mark applied by an instructor will lose marks for the reviewing student.

As well as judging correctness levels for query answers, reviewing students are also required to leave a comment. Students are encouraged to give comments of praise or positive suggestions for improvement. This is arguably the most valuable part of the reviewing process for both the reviewer and the reviewee.

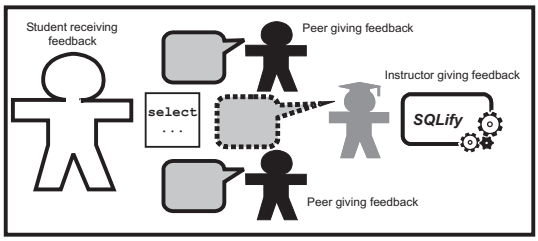

Figure 5: Feedback received by the student.

For the reviewer this is an opportunity to evaluate the work of a peer and in doing so, reflect on their own work. This requires higher order thinking skills [2] which will hopefully encourage greater learning outcomes.

For the reviewee receiving peer feedback means they will receive feedback from more sources than just the instructor or the system. The information contained in comments can encourage a more personal relationship among students (even anonymously) and between instructors and students, thus helping to form a community of learners.

For instructors, adding a comment allows elaboration on why a student may have lost marks and positive encouragement on their progress. The instructor may draw on a list of previously created comments to speed up the moderation process. This also provides consistency when multiple instructors are performing moderations.

Another benefit of this system is to allow students to flag peer reviews they believe to be incorrect for instructor intervention. Although quite often the instructor would be moderating such cases, this feature allows the student to express unhappiness with a review. This can remove some anxiety related to having their work assessed, in part, by algorithms and peers.

Note that the manner in which *SQLify* uses peer review is based on an existing, fully evaluated and successful peer-review system [6, 7].

3.1.3 Receiving Feedback and Marks

When all reviews of a student's work are complete, the instructor allocates a mark for the student's work based on the levels suggested by the *SQLify* system (which does so on the basis of its own algorithm and marks suggested by peers). Instructors must attend to submissions that have been assessed differently by each peer or by the system. Past experience [7] has shown that in at least half of normal submissions, peers alone are able to achieve non-conflicting reviews, so this means moderation is most likely to be unnecessary in these cases. Additionally, in many cases *SQLify* can determine a level for a solution with absolute certainty so this further eases the marking load of the instructor.

Past experience also suggests that it is important that students sense the instructor's involvement in the assessment process [7]. They see the instructor as an authority and feel they deserve the attention of the instructor during the assessment process. It is possible for good students who produce excellent work, to be assessed equally by peers and the *SQLify* system. In such cases the instructor may elect to assign a mark based on the agreed standard of the work without performing moderation. If a student achieves this consistently through the semester, they may miss the instructor's input in their assessment; they may then feel cheated by the assessment approach. It is possible to track how many times a student has been moderated by an instructor and set target levels of moderation at various points through the teaching period. This way each student can be satisfied with the attention they are receiving while still reducing the marking load on instructors.

One of the clearest benefits of using a single-step peer review system it that students receive feedback about their submission as soon as a peer has completed their review. Compared with a normal instructor marked assignment where students must wait until after the assignment deadline for feedback, previous use of the approach suggested here returns feedback to students within hours [7].

Once the peer review process is completed and the instructor has assigned marks to students, the *SQLify* system can calculate a final mark for each student.

The system suggests a final mark for a student's assignment. It does so by summing both the correctness marks for each query answer and accuracy marks for the reviews conducted by that student (see details in

Section 3.2.1). The weighting of correctness and review accuracy for each problem in each assignment could be varied according to the effort for each. An example would be weighting the correctness marks to 70% of the entire assessment and review accuracy marks to 30%.

The instructor then chooses to accept or modify the suggested mark (see Section 3.2.2). Such marks may be released individually by the instructor or *en masse*.

3.2 Details of how a correctness mark is determined

The three phases described above result in a process of assigning marks to queries and reviews. Three variables are kept per submitted query answer for each student: a system correctness mark sys, and two correctness marks from peers std1 and std2. With the aid of *SQLify*, the instructor then uses these marks to determine an overall *correctness mark* for each query answer the student submits.

3.2.1 How the system determines its correctness mark (Determining values for sys)

The levels below are taken from Figure 3.

L0 The submission is syntactically incorrect
The submitted query is sent to the database engine which returns a syntax error. The system is *certain* that the query is syntactically incorrect, so the internal value for sys is L0.

L1,L2 The submission is syntactically correct
The query is accepted by the database, upon which the system checks whether the output schema is the same as the one produced by the solution query (supplied by the instructor). The system can determine this exactly, and assigns an internal mark of L1 for sys if the condition is not met, and L2 if the condition is satisfied.

L6 The submission produces a result that is probably correct but needs to be checked or compared with peer marks
The query passes the output schema test, and now undergoes examination of its semantics. If the query does not belong to the Conjunctive Query (CQ) class, only a heuristic approach is possible. If the heuristics determine that the query is correct, there is only a small chance that in fact the query is not semantically correct (see Section 2.1). Hence, the internal value for sys is set to L6 if the test is successful, and L2 if it is not.

L7 The submission is certainly correct
In case the query belongs to the CQ class, it is possible to algorithmically decide whether it is semantically equivalent to the set solution query. If it is, the internal value for sys is set to L7, otherwise sys will be reset to level L2.

As is clear, there is a significant gap between levels L2 and L6; levels L3 to L5 cannot be chosen by the system. This is because the *SQLify* system cannot determine how good or how bad a query is that has been proven to be semantically incorrect. Hence a combination of peer review and instructor intervention is used to come up with a wider range of accuracy marks. Thus, as well as enhancing the learning experience of students, the peer review process also plays a practical role in moderating the mark proposed by the system and in flagging possible problems to the instructor.

3.2.2 How the instructor determines a correctness mark for an answer

SQLify calculates a suggested correctness mark for each submitted answer by using the marks given in reviews by students when its own automatic assessment is not sufficient. In many cases the system can suggest a mark with great certainty which the instructor can accept. Instructor moderation is needed when the system is uncertain about the student's submission or if there are conflicts between peer reviews or between reviews and the mark of the system. The following procedure is used by the instructor to apply the mark suggested by *SQLify*.

$\mathsf{sys} \leq \mathrm{L1}$
(The submission is incorrect)

In this case, the suggested mark will be same as sys, so possible internal values for the correctness mark are L0 or L1. The system is always right in these cases, so no student review marks need to be used to determine the correctness mark and no instructor intervention is necessary.

$\mathsf{sys} = \mathrm{L2} \wedge \mathrm{L2} \leq \mathsf{std1} \leq \mathrm{L4} \wedge \mathrm{L2} \leq \mathsf{std2} \leq \mathrm{L4}$
(The submission is largely incorrect)

In this case, both reviewing students agree with the system that the submitted query is semantically incorrect while the syntax and output schema are correct. The suggested mark will be the average of both student reviews rounded up to the nearest level. So, possible internal values for the correctness mark are L2, L3 and L4 as chosen by the instructor.

$\mathsf{sys} = \mathrm{L2} \wedge \neg(\mathrm{L2} \leq \mathsf{std1} \leq \mathrm{L4} \wedge \mathrm{L2} \leq \mathsf{std2} \leq \mathrm{L4})$
(There is a conflict between reviewers and the system)

In this case either one or both of the reviewing students disagree with the system. Hence, instructor intervention is appropriate to moderate the conflict. The correctness mark will be determined by the instructor and can be taken from L2, L3, L4, or L5 as suggested by the system. The correctness mark cannot be higher than L5 because *SQLify* has determined the query to be semantically incorrect. Also, the instructor can choose from the two additional levels L3 and L5 because he is more experienced than the reviewing students.

$\mathsf{sys} = \mathrm{L6} \wedge (\mathsf{std1} \leq \mathrm{L4} \vee \mathsf{std2} \leq \mathrm{L4})$
(The system suggests the answer is probably correct but the reviewers disagree)

The system could only heuristically determine that the query semantics are likely to be correct, while at least one of the reviewing students believes that the query is incorrect. In this case, intervention is needed by an instructor, who may choose any of the suggested levels L0, L2, L6, and L7. The chance is rather low that the query is indeed incorrect, so the first two levels are only used when the instructor believes that the student may have attempted to cheat (L0) or only accidentally confused the system (L2).

$\mathsf{sys} = \mathrm{L6} \wedge \mathsf{std1} \geq \mathrm{L5} \wedge \mathsf{std2} \geq \mathrm{L5}$
(The system thinks the query is probably correct and the reviewers agree)

Both students believe the query may be correct, and *SQLify* has also determined that this is likely. Intervention is not necessary as students are motivated to conduct accurate reviews. The system suggests a correctness mark of L7 (100%).

sys = L7

(The system indicates that the answer is certainly correct)

The system has incontrovertibly determined that the submitted query is correct, hence no student review marks are needed to determine the correctness mark, and no instructor intervention is necessary. The system strongly suggests a correctness mark of L7 (100%).

Based on the system's recommendation the instructor will set a correctness mark for each submitted answer. Each correctness mark is summed together with the marks calculated for the accuracy of reviews submitted by a student to form a final mark which can be released to the student.

4 Examples

To illustrate the workings of *SQLify*, two query problems are presented together with a description of how they are evaluated using *SQLify*. The assessment process to see how students' submissions are evaluated is followed through to a final mark.

4.1 Query Problems

The example problems make use of a database with the following schema[5].

employee(eNo, fname, lname, wage, dNo, eloc)
department(dNo, dname, dlocation)

The first query problem (QP1) is an example of a Conjunctive Query (a problem in class CQ). In this class it is possible to conclusively determine if a supplied query is correct without employing heuristic comparison.

Give the first and last names of all employees in the Sales department earning more than 300 dollars.

The instructor supplies a solution query that will be used by the system to test queries submitted by students.

```
SELECT fname, lname FROM employee E,
department D WHERE E.dNo = D.dNo AND
dname = 'Sales' AND wage > 300;
```
(QP1)

The following are two queries submitted by students. They are both different to the solution presented by the instructor, but both can be proved to be semantically equivalent to the instructor's solution query and are therefore considered correct. Correctness marks given by the system and two peers are also shown.

Submitted query	sys	std1	std2
`SELECT fname, lname FROM employee JOIN department ON dNo WHERE dname = 'Sales' AND wage > 300;` (SA1)	L7	L6	L7
`SELECT fname, lname FROM employee E WHERE wage > 300 AND EXISTS (SELECT * FROM department D WHERE E.dNo = D.dNo AND dname = 'Sales');` (SA2)	L7	L7	L4

[5]Instructors submitting their own query problems can submit their own schemas and instances.

The following query is an incorrect query answer to the above problem (QP1).

Submitted query	sys	std1	std2
`SELECT fname, lname FROM employee E WHERE dname = 'Sales' AND wage > 300;` (SA3)	L2	L6	L4

The next problem (QP2) involves a query that is not in CQ class.

List all locations where there is either an employee or a department.

The following is an instructor's solution query for this problem.

```
(SELECT eloc FROM employee) UNION
(SELECT dlocation FROM department);
```
(QP2)

An incorrect solution to this problem is given next.

Submitted query	sys	std1	std2
`SELECT loc FROM employee, department WHERE loc = eloc OR loc = dlocation;` (SA4)	L2	L2	L3

4.2 Marking query correctness

When the system has evaluated a submitted query and peer reviews are complete for that query the system will recommend a mark to the instructor. The instructor can then assign a *correctness mark* for the query.

The table below shows, for each row, the correctness marks for a particular query submitted by a student, as given by the system itself (sys), and two peers reviewing the query answer (std1 and std2). In addition, a suggested mark is shown calculated by *SQLify* on the basis of sys, std1 and std2 using the procedure described in Section 3.2.2. Finally, the correctness mark assigned by the instructor is listed; this mark may or may not be the same as the suggested mark.

Student	Problem	Submitted query	System mark (sys)	Reviewer 1	Mark (std1)	Reviewer 2	Mark (std2)	Suggested mark	Correctness mark
1	QP1	SA1	L7	3	L6	5	L7	L7	L7
1	QP2	SA4	L2	4	L2	5	L3	L3	L3
				...					
4	QP1	SA2	L7	1	L7	3	L4	L7	L7
5	QP1	SA3	L2	1	L6	2	L4	L4	L4

The internal values corresponding to levels given in Figure 3 are not hard-coded into the system. The instructor using *SQLify* can set these values during use of the system. Hence the levels given in the last column will translate into different scores for queries as determined by the instructor.

4.3 Checking accuracy of reviews

The following table lists one row per peer review that is performed in the context of an assignment. The first row, for instance, shows that student 1 was a reviewer for a query (SA2) submitted by student 4 in answer to query problem QP1. Student 1 gave this query answer a correctness mark of L7. The correctness mark for the submitted query answer ultimately

given by the instructor was also L7. Hence, the *accuracy mark* for this particular review is 100. For the next review performed by this student there is a difference between the correctness mark given by this student and the correctness mark set by the instructor. This difference causes their mark for accuracy to be reduced.

Reviewer	Reviewee	Problem	Submission	Reviewer's mark	Correctness mark	Difference	Accuracy mark
1	4	QP1	SA2	L7	L7	0%	100%
1	5	QP1	SA3	L6	L4	20%	80%
				...			

4.4 Calculating a final mark

The last table below summarizes the various marks that a particular student received for various query problems and for the reviews performed. A weighted final mark is given in the last row using the suggested weightings of 70% for correctness and 30% for accuracy of reviews.

Student: 1		
Correctness marks	QP1	100%
(Weight 70%)	QP2	50%
	QP3	70%
Review accuracy	QP1	100%
(Weight: 30%)	QP2	80%
	QP3	50%
Final Mark		74%

5 Implementation Aspects

The *SQLify* system has been implemented and is currently undergoing tests to prepare for first use in an undergraduate database systems course offered later this year.

The current version of the system was implemented in a standard LAMP (Linux, Apache, MySQL, PHP) environment and is integrated with university systems to manage assignment assessment for enrolled students. However, the tutoring part of *SQLify* is open for outside use, and can be accessed at [8].

While there are several implementation issues that are worthy of description, we here only focus on the process of testing query equivalence.

5.0.1 Heuristic Testing

When students submit a query for assessment, *SQLify* first rewrites it to counter SQL injection attacks and to align it with the actual database table names stored in MySQL. As will become clear, the rewriting is also useful for other reasons. Subsequently the system checks syntactic correctness and the correctness of the query's output schema. If either of these fail, evaluation stops and sys is set to either L0 or L1.

After rewriting, the heuristic test given in Formula 1 on page 2 is performed. The same measures used in *AsseSQL* and *SQLator* are taken to increase the reliability of the heuristic test.

If the heuristic test is negative, meaning that for the specific database instance or instances stored in MySQL the result of the query is different from that of the correct query (as supplied by the instructor), evaluation stops and the submitted query's sys value is set to L2.

In case the test was positive, but queries involve set operations, grouping, or aggregation, the analysis stops, and *SQLify* sets the sys variable to L6. Peer reviews and instructor input will then further refine the score for the submitted query.

5.0.2 CQ Equivalence Testing

If the submitted query is in the Conjunctive Query class automatic evaluation continues. This is the case if the query was able to be rewritten (in the previous phase) into the following SQL form:

SELECT $A_1, \ldots, A_n$
FROM $\text{table}_1, \ldots, \text{table}_t$
WHERE $\langle condition_1 \rangle$ and ... and $\langle condition_c \rangle$

where the $condition_i$, for $1 \leq i \leq c$, consists of only join conditions and equality comparisons, that is $X = Y$, where X is a variable (column) and Y is either a variable or constant.

Note that some SQL queries containing WHERE EXISTS subqueries can be rewritten in this form. Query SA2 given in Section 4 is such a case, and can be rewritten as QP1 which conforms to the SQL subset given above.

Testing the equivalence of conjunctive queries is a basic problem on which query optimizers are partly based. Since checking equivalence of two queries can be done by testing the mutual containment of the queries involved, the containment problem has been studied extensively by many researchers.

Under the set semantics, the containment of conjunctive queries using only equality tests was fully solved by Chandra and Merlin [4], using the concept of containment mapping. There has been extensive work on the testing of set containment of inequality conjunctive queries, and also on bag semantics with either equality or inequality tests.

Recently, the idea of using a finite set of canonical databases to represent an infinite set of databases is used by Penabad [15] to develop a general procedure, called Query Containment Checker (QCC), to test the containment problems of both equality and inequality conjunctive queries, under both set and bag semantics.

For *SQLify* we decided to implement an algorithm using tableaux representation of expressions [1] to test the equivalence of equality conjunctive queries under set semantics. The tableaux can be easily constructed from the previous query rewriting phase. We will leave all other fragments of the general containment problem as an extension for future versions of our system.

6 Conclusion and Future Work

In this paper a small set of existing tools used for teaching and assessing SQL writing skills was reviewed. The tools were evaluated both from Computing Education and Database perspectives, noting possible areas of enhancement.

Secondly, we proposed a comprehensive new tool for the teaching and assessment of SQL writing skills. Central to the system is the use of an intricate automatic grading system and peer review. The main reason for including peer review is to offer students a richer learning experience. Additionally, peer reviews assist in the assessment of assignments.

SQLify uses a relatively complex method to assign final grades to assignments, designed to (1) yield a much wider range of grades than simply *correct* or *incorrect*, (2) utilize database theory to arrive at a computer assisted assessment, (3) set high quality demands for student reviews, yielding a better learning environment, and (4) reduce the number of necessary interventions performed by course instructors.

Regarding future work, we are currently preparing *SQLify* for use in a live course by the end of 2006. We will then evaluate the usefulness of the system as perceived by students and instructors. Any change in student outcomes will be measured.

To evaluate relational algebra expressions two additions to the system (planned for the next version of *SQLify*) are needed: first, the implementation of an interface that helps students construct syntactically correct algebra expressions, and second the implementation of the algorithm that translates the submitted algebra expression to an equivalent SQL statement. The generated statement is then processed in the same way as a normal SQL statement.

References

[1] Serge Abiteboul, Richard Hull and Victor Vianu. *Foundations of Database Systems*. Addison Wesley, 1997.

[2] B. Bloom. *Taxonomy of Educational Objectives*. Edwards Bros., Ann Arbor, Michigan, 1956.

[3] C. Brook and R. Oliver. Online learning communities: Investigation a design framework. *Australian Journal of Educational Technology*, Volume 19, Number 2, pages 139–160, 2003.

[4] Ashok Chandra and Philip Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90, Boulder, Colorado, 1977.

[5] Thomas Connolly and Carolyn Begg. *Database Systems – A Practical Approach to Design, Implementation, and Management*. Addison Wesley, fourth edition, 2005.

[6] Michael de Raadt, Mark Toleman and Richard Watson. Electronic peer review: A large cohort teaching themselves? In *Proceedings of the 22nd Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education (ASCILITE'05)*, pages 159–168, Brisbane, December 2005.

[7] Michael de Raadt, Mark Toleman and Richard Watson. An effective system for electronic peer review. *International Journal of Business and Management Education*, Volume 13, Number 9, pages 48–62, 2006.

[8] Stijn Dekeyser, Michael de Raadt and Tien Yu Lee. *SQLify* project website. Technical report, 2006. `http://www.sci.usq.edu.au/projects/sqlify/`.

[9] Suzanne Dietric, Eric Eckert and Kevin Piscator. WinRDBI – a Windows-based relational database educational tool. In *Proceedings of SIGCSE '97*, pages 126–130, San Jose, California, March 1997.

[10] R. Kearns, S. Shead and A. Fekete. A teaching system for SQL. In *Proceedings of ACSE '97*, pages 224–231, Melbourne, July 1997.

[11] Claire Kenny and Claus Pahl. Automated tutoring for a database skills training environment. In *Proceedings of SIGCSE'05*, pages 59–62, St. Louis, Missouri, February 2005.

[12] Michael Kiefer, Arthur Bernstein and Philip Lewis. *Database Systems – An Application-Oriented Approach*. Addison Wesley, second edition, 2006.

[13] J. Kurhila, M. Miettinen, P. Nokelainen, P. Floreen and H. Tirri. Peer-to-peer learning with open-ended writable web. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, pages 173–178, Thessaloniki, Greece, June 2003.

[14] Antonija Mitrovic. Learning SQL with a computerized tutor. In *Proceedings of SIGCSE'98*, pages 307–311, Atlanta, Georgia, February 1998.

[15] Miguel Penabad. *General Procedure to Test Conjunctive Query Containment*. Ph.D. thesis, Universidade da Coruña, 2002.

[16] Julia Prior and Raymond Lister. The backwash effect on SQL skills grading. In *Proceedings of ITiCSE'04*, pages 32–36, Leeds, UK, June 2004.

[17] Shazia Sadiq, Maria Orlowska, Wasim Sadiq and Joe Lin. SQLator—an online SQL learning workbench. In *Proceedings of ITiCSE'04*, pages 223–227, Leeds, UK, June 2004.

[18] D. Saunders. Peer tutoring in higher education. *Studies in Higher Education*, Volume 17, Number 2, pages 211–218, 2006.

[19] Ben Shneiderman. Improving the human factors aspect of database interactions. *ACM Transactions on Database Systems*, Volume 3, Number 4, pages 417–439, 1978.

[20] Abraham Silberschatz, Henry Korth and S. Sudarshan. *Database System Concepts*. McGraw-Hill, fifth edition, 2006.