Automated Assessment for Databases Units

Anthony Kleerekoper, Andrew Schofield
Department of Computing and Maths
Manchester Metropolitan University
Manchester, UK
a.kleerekoper@mmu.ac.uk

ABSTRACT

With ever-growing class-sizes, automated assessment can be an extremely valuable tool in higher education. In this paper, we present our tool for automatically assessing SQL programming and reflect on five years of its use. We highlight some of the changes and challenges we have encountered as well as lessons learned. Our tool has proven successful in both its primary goal and in secondary goals such as encouraging student participation. Since its inception it has grown incrementally and been adapted for other contexts. It is now undergoing a major overhaul to expand its remit to include elements of database design and theory. We will discuss how this is being done and how we are aiming to construct a single, integrated assessment tool. Ultimately, the tool could be adapted to other contexts as well and our aim is to raise awareness of the issues facing automated assessment and encourage its adoption.

KEYWORDS

Automated Assessment, Databases, SQL, Web-based Learning

ACM Reference Format:
Anthony Kleerekoper, Andrew Schofield. 2024. Automated Assessment for Databases Units. In *Computing Education Practice (CEP '24), January 05, 2024, Durham, United Kingdom.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3633053.3633059

1 INTRODUCTION

Large classes have been a problem for many years in Computer Science Education and are only likely to increase. One of the challenges it poses is how to provide authentic assessments with timely feedback. Automated assessment systems are a frequently used solution.

In this paper, we describe our automated assessment tool for SQL and reflect on our experiences of using it and improving it over five years. SQL Tester was introduced in the 2017/18 academic year to replace an existing non-automated assessment which had a number of serious problems. An initial experience was extremely positive and the first version was reported on in [2].

SQL Tester has proved popular with students over many years and in different teaching units. It has been successful in meeting its primary aim of providing authentic assessment at scale. By incorporating practice tests, it has also enabled and encouraged students to revise more and this has led to a slow expansion of its learning functionality. It is currently evolving into *Mmudbox*, MMU's database sandbox, which also support multiple-choice questions.

Our experience with SQL Tester is that the initial time invested in producing a bespoke automated assessment tool has been extremely valuable over many years and has led to improved educational outcomes for many cohorts.

CEP '24, January 05, 2024, Durham, United Kingdom

ACM ISBN 979-8-4007-0932-6/24/01.
https://doi.org/10.1145/3633053.3633059

2 MOTIVATIONS AND USAGE CONTEXT

The creation of SQL Tester was motivated by the needs of a second-year undergraduate unit called Database Systems. The unit was originally taught over 24 weeks to approximately 120 students per year. The students had previously taken a first-year unit that covered some database elements. When we took over the unit, 10% of the unit grade came from a set of 5 portfolio tasks that required students to write SQL Select statements. These tasks were marked during lab time by the lab tutors.

We found that students coming from the first-year unit often did not have a strong grasp of SQL and the portfolio tasks were not encouraging them to learn. Since these tasks were completed and marked during lab-time, collusion was the norm and little attention was paid to learning SQL via labsheets. Moreover, much of the tutor time was taken marking and not helping.

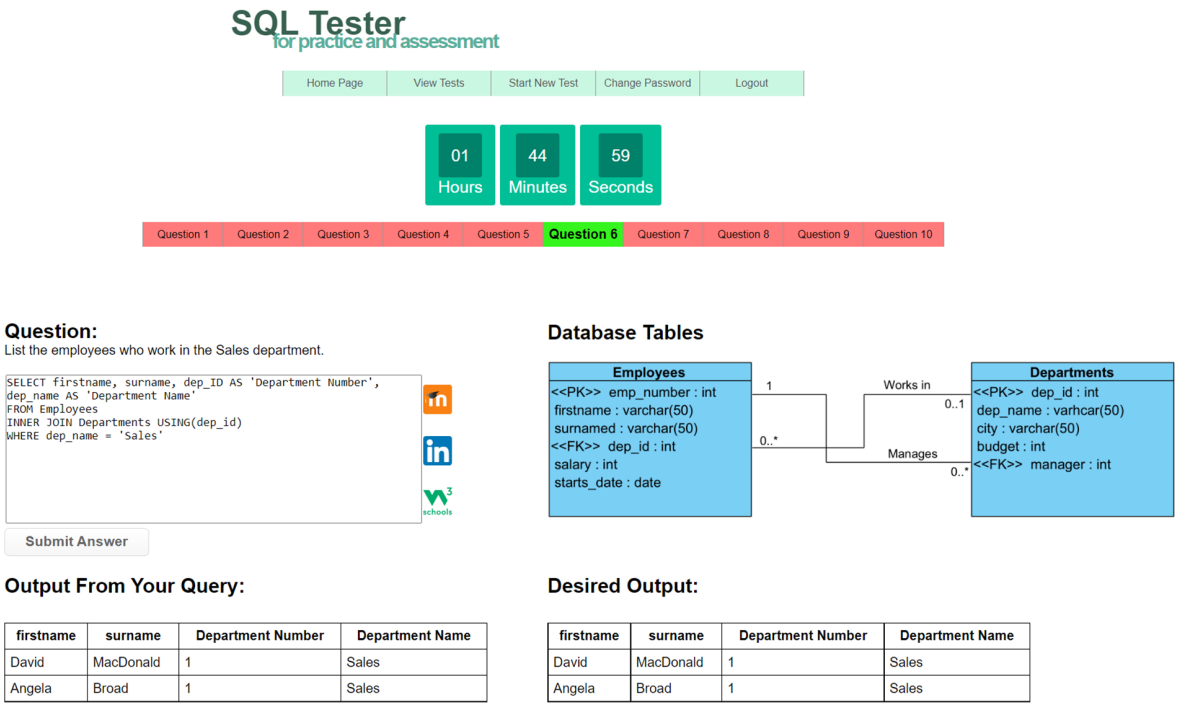
Therefore, in the Summer of 2017 we decided to adopt an alternative, automated approach. From the literature, we identified AsseSQL, a tool produced by Prior *et al.*, as an archetype system [4, 5]. Unfortunately, the source code for AsseSQL was not publicly available (and is still not to the best knowledge of the authors), nor did we receive a response from the authors. Therefore, we developed our own version of AsseSQL and called it SQL Tester. The main components of SQL Tester were essentially the same as those described in AsseSQL.

SQL Tester was used for a five years in its original unit and was also used in another, smaller unit of about 20 Degree Apprentice students. During the Covid-pandemic, the size of the tests were increased and the weighting increased to 30% because another element of the unit's assessment (a group-based coursework) was no longer considered usable.

Due to changes in our institution's degree courses, the original unit no longer runs and instead there is a first-year unit that focuses solely on databases. SQL Tester is now used in that unit to provide 50% of the unit grade (this unit is also worth 15 credits whereas the second-year unit was worth 30 credits). This unit is taught to about 450 students per year.

3 SQL TESTER FUNCTIONALITY

SQL Tester presents students with a timed test where their answers are automatically and immediately marked. A typical test consists



firstname	surname	Department Number	Department Name
David	MacDonald	1	Sales
Angela	Broad	1	Sales

firstname	surname	Department Number	Department Name
David	MacDonald	1	Sales
Angela	Broad	1	Sales

Figure 1: A screenshot of the SQL Tester system during a practice test.

of 10 questions to be completed within 50 minutes (though when the test was worth 30% of the second-year unit we increased this to 20 questions in 100 minutes). The questions can be attempted in any order and as many times as a student wants. The test continues until either the time runs out or all questions are answered correctly (or the student gives up trying). Each test has its own *base length*, but when students are entered into the system they can be given a multiplier to give them more time. This is used to comply with Personal Learning Plans that require appropriate adjustments for students with personal needs.

Alongside each question, the students are shown an Entity Relationship Diagram (ERD) of the database the question relates to. The ERD shows the table names; the column names and data types; and illustrated foreign key relationships. Antonija Mitrovic suggested that having to memorise the database schema was a contributor to student errors [3]. Therefore, SQL Tester includes the ERD alongside every question to reduce this burden.

Below the ERD, the student sees the desired output based on a model answer that the tutor has set for their question and a set of sample data. The student can use this to help guide them to the correct answer. Although this is not authentic because in the real-world a developer won't typically know exactly what the output of a query should look like, Prior *et al.* argued that including this helps remove ambiguity and is especially helpful for students for whom English is not their first language [5]. This is a persuasive argument.

Immediately below the question text, there is a textbox for students to type their solution. Every attempt is logged by the system and when a student returns to a question they have already attempted their last attempt is shown in the textbox. This allows students to save some of their progress and makes it easier for them to come back to a previously attempted question.

Finally, when they submit an attempt they are shown the output from their query. If their submission contained a syntax error, the error message from the database system is shown. If there were no syntax errors, the output from their query is shown. Whatever the output, if their solution was not correct they can use the output to help guide them towards the correct solution.

If their attempt was correct, the appropriate question number in the list turns green to indicate it is correct. If not, a message box appears to inform them that they are wrong and encourage them to keep trying. We only use a message box for incorrect answers because otherwise the student may not realise that their attempt was submitted (as the output might not obviously change from their previous attempt).

It is worth pointing out that although we did not strongly consider colour-blindness when choosing the colours for the question backgrounds, we have never been told by any students that they struggle to differentiate between correct and incorrect questions. A check of the colours from a screenshot using the color-blindess simulators Coblis and Pilestone also show that the colours can be distinguished even by those with colour blindness.

3.1 Question Banks and Categories

Every test in SQL Tester consists of questions drawn at random from a question bank. We aim to have a large bank of questions for each test so that students can attempt the same practice test many times. Each question within the bank belongs to a specific category and tests are created by drawing questions at random from the categories in a set proportion. Initially, we had nine categories (with

two questions drawn from the *Inner Join* category). The categories chosen were influenced by those used in AsseSQL (as reported in [1]), but with some changes. Specifically, we dropped the two categories they found to be the hardest for students (self-joins and correlated subqueries) and added some simpler ones (such as questions requiring a SELECT with a row function). In the most recent version we have simplified many of the questions and added a *Challenge* category to stretch the strongest students.

3.2 Marking

In the original version, an answer was marked as correct only if it matched the desired output precisely. That meant it had to have the same column names and same row contents in the same order and case. We required rows to be in the same order because some of the questions required a specific order. We also required the cases to match because some questions required specific cases.

One of the big problems with this method of marking was that it was possible to trick the automated marker by producing the desired output without correctly answering the question. For example, some questions required the use of a subquery to provide the correct filter in a WHERE clause. However, it was possible to produce the correct output without using a subquery by noting the desired output. In the first year, we manually checked through all answers to correct for these occurrences.

In the second version, we introduced a second, hidden set of tables with different data. Student submissions were compared not only against the visible data but also against the hidden data. Only if their outputs matched for both sets of data was their answer marked as correct. This prevented them from gaming the marking system. For the first iteration of this system, we only used the hidden tables to flag a submission as possibly being incorrect and continued with manual checking. However, after the first year of using this we felt confident that the second system worked and so incorporated it fully.

When a student submits an answer that produces the correct output on the visible data but incorrect on the hidden data, they receive a message to that effect. Currently, this message reads, "Unfortunately that answer was not correct. It produces the desired output for the sample data, but does not answer the question. Keep trying!" Some students do not fully understand the message but we have always been able to explain it to them.

We have also softened our marking criteria over the years. In the latest version we no longer use case-sensitive tests for the data because we have switched to teaching MariaDB which, by default, is mostly case-insensitive. As a result, we removed questions that required using the UPPER or LOWER functions.

Furthermore, the requirement that the output be in the same order as the desired output has generally been dropped. The exception is of course when the question specifies an order.

The final big change in marking is with regards to column names. Initially, these had to match the desired names exactly meaning that if a column had an alias in the model answer, it had to have the same alias in the output. We needed to include aliases in model answers to hide function calls. However, we felt that requiring aliases to be correct to get the answer correct was too harsh. It effectively meant that students who did not know how to make an alias would very likely fail the entire test. For a period, we stopped checking column names entirely and only checked that the content was the same. But this meant we could never test a student's ability to create aliases.

Figure 2: Three question buttons showing the different states. Question 1 is currently incorrect, Question 2 is correct but with incorrect column names and Question 3 is completely correct.

In the latest version we have changed our marking system so that a question can have a partial mark (whereas previously an answer was either right or wrong). Now if an answer is completely correct, including the column names, then the submission has a mark of 1. However, if the column names are wrong, the submission has a mark of 0.9 (though this value can be tweaked as desired). A correct answer with incorrect column names is indicated with diagonal red and green lines which show it is partially correct. The three backgrounds of question buttons is shown in Figure 2, where Question 1 is incorrect, Question 2 is correct with incorrect column names and Question 3 is completely correct.

4 STUDENT FEEDBACK

Our own observations are that students enjoy using the SQL Tester and engage well with it. The more motivated students complete the topic tests as they progress, but even the less motivated take many practice tests in the run-up to the assessed test. In end of unit feedback surveys we receive positive comments about the tester. For example, in the latest end of unit survey, one student commented: "The SQL tester was extremely helpful for revision, couldn't recommend it enough." Another student sent the following in an email to tutors:

> "[SQL Tester] was by far the best due to it allowing students like myself and others the ability to see where about they are at in terms of sql confidence and I personally enjoyed this because it allowed me to see how my progress went from little to no confidence to then being ever so confident "

5 FROM ASSESSMENT TO TEACHING TOOL

SQL Tester started purely as an assessment tool. The first version did include practice tests so that students could familiarise themselves with the test system before the assessed test. The practice tests offered a different scenario than the assessed test and therefore, obviously, different questions. But the questions were drawn from the same categories and in the same proportions as the assessed test and we tried to make the level of difficulty about the same.

From the beginning we observed that students engaged very strongly with the practice tests, far more than we anticipated. Students took a median of 8 practice tests each before their assessed test and, on a voluntary questionnaire, more than 90% agreed that they wanted to keep taking practice tests until they could get a good mark.

After a couple of years, we added five "topic tests" that aligned with the SQL topics we taught and offered more focussed practice

in addition to the existing practice tests. We encouraged students to take the topic tests as a final-step in their weekly learning, after completing the labsheets.

Since SQL Tester is now used for a first-year unit we have increased our steps towards expanding the role of SQL Tester into a teaching aid not just an assessment tool.

As a first step, help links were added to the tester for the topic tests. These can be seen in Figure 1 as three icons immediately to the right of the answer box. The aim of the links was to provide students with a way of finding more direct help with a particular question. Since this was experimental, the help links were not enabled by default and students had to opt-in to them being available for their particular instance of a topic-test.

We offered three links for each question. The first was a link to a unit "handbook" (more akin to a bespoke unit textbook) which took students to the relevant page where they could read about the function or concept required for answering the question. The second took students to a LinkedIn Learning [1] video relevant to the question (as the University has a subscription to LinkedIn Learning and all students have access to it). The third link was to a relevant w3schools page [2].

Although a full analysis of the use of these links is still ongoing, our initial observations were that most students did not use them. One of the challenges with the help links was trying to predict what concept would most help the students, since in any given question there are multiple concepts required. We didn't always get this right so many times the links were not actually helpful to students. This is an area we are actively working on.

SQL Tester is now undergoing a major overhaul and evolving into Mmudbox (MMU's database sandbox), an educational sandbox to aid the teaching of databases, both SQL programming and design theory. As part of that evolution, we are incorporating new elements into the SQL side. For a start, help links will always be available to students in their topic tests.

We are also adding an option for students to show a partial answer for the topic tests. The partial answer will show them the structure of the answer and explicitly show where some SQL is missing. For example, where a function is required or that another clause is needed. Additionally, we will be making model answers visible to students once their test ends so they can see how the question could have been answered.

A major component of the overhaul is that we will be incorporating multiple-choice questions into Mmudbox, both single-response and multiple-response questions. Previously, we were running a separate multiple-choice question test using Moodle, but we wanted to integrate everything into one system. These questions will be used to assess students' understanding of database design concepts and their understanding of ERDs. By adding practice multiple-choice question tests into the system, this will hopefully encourage more revision and preparation as we observed with the SQL questions.

One area we haven't yet tackled, but have plans to, is with the error messages from the database management system. We find, consistently, that students do not make good use of these messages to help them solve syntax errors when writing SQL. Often this is because the messages are too vague to be helpful, or point to the wrong part of the code. However, sometimes the messages are clear enough but not to students. So, we will be looking to add some explanation to the message and a hint based on it. For example, when the error message states that a table "doesn't exist" we may add a note that explains to the student that they most likely have a spelling or case error in their FROM clause.

6 CONCLUSION

SQL Tester is an automated assessment tool for SQL that has been used and improved for five years. It is now evolving into a more general-purpose database assessment and teaching tool which we are calling Mmudbox (MMU's database sandbox). The original idea behind the Tester was to replace a small assessment that consisted of portfolio tasks that were marked in lab sessions and had a number of disadvantages.

Student engagement with the tool has been very strong and they have used it as a learning tool by testing themselves with the practice tests and later the added topic tests. Feedback from students has been very positive.

SQL Tester was not a unique concept in its original form, as it was based heavily on the description of AsseSQL. However, its new evolution is the result of a move from being an assessment to a teaching tool with more support for learning SQL, including (a) links to helpful sources for each question (b) the ability to see a partial answer for practice tests and (c) being able to see a model solution when the time is completed.

Mmudbox will be a more general-purpose tool that includes multiple-choice questions that will be used to assess database design and other concepts. This replaces an existing test that was run in Moodle and brings all our database assessments into one place.

Our experience indicates that providing students with a safe, simple and responsive environment to practice SQL programming encourages more practice and engagement. Students enjoy the instant feedback, even if its only correct/incorrect and can gain confidence from this. We believe that this approach can be more widely applied in other contexts and institutions to provide scalable and authentic assessment systems.

REFERENCES

[1] Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. 2015. A Quantitative Study of the Relative Difficulty for Novices of Writing Seven Different Types of SQL Queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '15*. 201–206. https://doi.org/10.1145/2729094.2742620

[2] Anthony Kleerekoper and Andrew Schofield. 2018. SQL tester: an online SQL assessment tool and its impact. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2018*. ACM Press, New York, New York, USA, 87–92. https://doi.org/10.1145/3197091.3197124

[3] Antonija Mitrovic. 1998. Learning SQL with a computerized tutor. *ACM SIGCSE Bulletin* 30, 1 (1998), 307–311. https://doi.org/10.1145/274790.274318

[4] Julia Prior. 2014. AsseSQL: an Online, Browser-based SQL Skills Assessment Tool. In *Proceedings of the 2014 conference on Innovation & technology in computer science education ITiCSE '14*. ACM Press, New York, New York, USA, 1. https://doi.org/10.1145/2591708.2602682

[5] Julia Coleman Prior and Raymond Lister. 2004. The backwash effect on SQL skills grading. *ACM SIGCSE Bulletin* 36, 3 (2004), 32. https://doi.org/10.1145/1026487.1008008

[1] https://www.linkedin.com/learning/
[2] https://www.w3schools.com/