Improving the Efficiency of Database-System Teaching

Jeffrey D. Ullman
Department of Computer Science
Stanford University
ullman@cs.stanford.edu

ABSTRACT

The education industry has a very poor record of productivity gains. In this brief article, I outline some of the ways the teaching of a college course in database systems could be made more efficient, and staff time used more productively. These ideas carry over to other programming-oriented courses, and many of them apply to any academic subject whatsoever. After proposing a number of things that could be done, I concentrate here on a system under development, called OTC (On-line Testing Center), and on its methodology of "root questions." These questions encourage students to do homework of the long-answer type, yet we can have their work checked and graded automatically by a simple multiple-choice-question grader. OTC also offers some improvement in the way we handle SQL homework, and could be used with other languages as well.

1. MOTIVATION

The cost of education has risen, relative to inflation, faster than almost any other industry. For example, in the past 35 years, tuition at major colleges, relative to the cost of sending a letter across the US by air, has increased fivefold. It has increased 5000 times relative to the cost of a cross-country phone call. Computerization has had little effect. There are some on-line courses to take, and there are some activities in the corporate-retraining arena. But nothing yet has reduced significantly the cost of college education. Our plan is based on the following principles:

- Residential college education cannot be replaced by impersonal, on-line courses. The social aspects of college, including learning in groups, informal teacher-student discussions, and so on, are too valuable and important.
- Savings must come from elimination of redundancy. At any time, there are hundreds of database courses being offered. They differ in many small ways. For instance, they have similar, yet distinct, homework assignments, each requiring substantial time to design, debug, and grade.
- By automating what can be replicated efficiently, we leave the course staff free to do the most important things: individualized help, group discussions, and the other aspects of the course that require attention from an instructor.

SIGMOD 2003, June 9–12, 2003, San Diego, CA.

2. WHAT CAN BE AUTOMATED?

I propose the following four items as good targets for automating aspects of a database course. Except for the last, the ideas carry over to other courses as well.

2.1 Lectures

I would like to see a repository of well-produced lectures, in modules of about half an hour, from which the instructor could select a consistent sequence. These materials would occupy about half the classroom time, with discussion or group problems occupying the other. In a sense, these materials are an extension of the textbook, which has for generations replaced the need for each instructor to develop their own class notes.[1]

2.2 Global Help Desk

A student with a problem (e.g., "The DBMS says 'semicolon required.' What does that mean?") needs immediate help. Having to wait for office hours means they must put their work away and resume it hours or days later. They should be able to send an email and get a response in a few minutes, at most. The technology to support this process already exists in on-line "call centers." You need to be able to route questions to an available TA, somewhere in the world, and you need to give the TA quick access to useful response information.

A back-of-the-envelope calculation tells us a one-semester course needs around 2000 students/year to occupy one TA 24/7. But no one TA can be on-duty 24/7, so we would need about 10,000 students/year to support a dedicated staff for that course. As a result, this sort of system is very hard to "bootstrap" into place. Yet it would be a wonderful and

[1]Some "not-well-produced" lectures consisting of Powerpoint with voiceover, were created by me and used in the Fall-quarter CS145 at Stanford. You can see them, in GIF format without the voiceover, at `www-db.stanford.edu/~ullman/dscb/gslides.html`. Slides with voiceover are available, but only if you agree to use (most of) them in a class.

cost-effective asset that could come into place through the proper level of investment.

2.3 Automated Homeworks and Exams

Systems capable of accepting on-line homework, and even of grading multiple-choice or short-answer questions, are common. Most of this article is devoted to what we see as an advance — the OTC "root question." In our approach, a relatively simple grading system is used to simulate the "expensive" form of homework, where students work long-answer questions, and their work is checked by the course staff.

2.4 Automated Labs

An important aspect of the database course is the access of students to a real DBMS. While large schools usually have such a facility available, and have the staff to maintain it, smaller institutions often do not. Bala Iyer at IBM created a system, NetDB2, that solves this problem, giving students access to DB/2 via a Web interface.[2]

But there is another way that SQL programming needs to be automated. I have often given students homework in which some schema is described, and then some English-language queries are stated. They are asked to write SQL queries to implement the English statements. I then give my poor TA's the job of staring at convoluted SQL queries to figure out what they do. One conscientious TA even created a database and typed the students' queries to see what they did. Judging by the number of students who have come to me complaining that what they wrote works even though the TA marked them wrong, this task is very hard to get right (curiously, no student ever complained that their query was wrong, yet the TA marked them right).

The OTC lab automates this process. Students are given the schema over the Web and asked to fill in boxes with certain queries. They can submit one or more of the queries as many times as they like, and are encouraged to keep working until all are right. If their query is syntactically incorrect, the error messages from the DBMS are reported to them. If the query is syntactically correct, it is run against a test database that is set up for this assignment only, and the result is checked for the presence and absence of certain values. While there is some chance that they will receive credit for a query that is incorrect, they will surely receive credit for a correct answer, no matter in what order they produce the attributes of the correct tuples. A future improvement is to provide a student with an incorrect query some indication of missing or inappropriate tuples. The problem is that, with a single test database and the right to resubmit many times, students may eventually learn too much about the test database.[3]

3. ROOT QUESTIONS

A *root question* is a multiple-choice question that has several right answers and many wrong answers. In this section, we shall show how root questions are used in OTC to simulate long-answer questions and encourage students to work the long-answer version of the family of multiple-choice questions that are entailed in a single root question.

3.1 Motivation for Root Questions

Our primary aim is to automate grading, yet give students the benefit of doing homework exercises and having their work checked. A secondary goal is to allow all students in a class to take a homework or exam on-line in a way that doesn't make collaboration easy. If we simply issue questions to be answered over the Web, one bright student will solve the problems and give the answer key to the others. Requiring that all students do the work at the same time has limited effect, since we cannot rule out collaborations occurring out of sight, e.g., using instant messaging. Requiring all students to appear at the same time to be proctored in a single room may be infeasible, since we then require a public machine for each student. Moreover, while exams may be handled this way, it is not realistic to proctor weekly homeworks.

One way to inhibit collaboration is to give each student different questions. However, each student must get similar questions, lest issues of fairness be raised. In some disciplines, such as Physics, it is easy to create a parametrized question, give each student a variant of the question with random values for the parameters, and have the system compute the correct answer as a simple arithmetic formula involving the parameters. Several such systems are in use.

EXAMPLE 3.1. *Here is an example of a question that can be implemented using arithmetic behind the scenes:*

> *If a rectangular room is $x feet by $y feet, how many square yards of carpet are needed to cover the floor completely?*

The system can select random integers for $x and $y and check that the student's answer equals $x \times $y/9. □

Database systems offers a few types of questions for which such a facility is useful (e.g., "A disk rotates at $r revolutions per second and has $b bits per track. How many $\cdots$ "). But most of the questions we would like to ask cannot be answered by simple arithmetic formulas. As a result, trying to adopt the "formula" approach to random-parameter questions would involve significant programming by the designer of each question.

EXAMPLE 3.2. *Here is a typical (but easy) question that might appear in database homework.*

> *If relation R has set of tuples $x and relation S has set of tuples $y, what is the natural join of R and S?*

Not only does the question designer need to generate random sets of tuples $x and $y, but they must implement a join algorithm — not an easy task. Additionally, the comparison of the result of this algorithm with the answer given by the student must take into account that the student may have listed the tuples of the result in any order and may have used various conventions regarding commas, braces, and other punctuation. An alternative is to explain and restrict the form of the answer precisely, but that makes the question itself lengthy and cumbersome. □

[2]H. Hacigümüs, S. Mehrotra, and B. Iyer, "Providing database as a service," *18th ICDE* (March, 2002), pp. 29–40.

[3]Roland Yap suggested that we explain errors in terms of a database isomorphic to the test database, thus concealing completely the values that appear in the latter.

3.2 How Root Questions Work

A root question about joins of relations would be phrased as follows. The *stem* (portion before the choices) of the question would give particular, small sets of tuples that constitute the relations R and S and then say: "which of the following tuples is in the natural join of R and S?" The question designer then gives OTC a list of possible correct answers. For this question, the list could consist of all the tuples in the natural join of these relations, or any subset of those, if the list were too long. The designer also gives a supply of incorrect answers, which in this case could be any other tuples. To make the *distractors* (incorrect answers) look plausible, it would be wise to give them the right number of components and to use values that appear in R and/or S.

While the OTC system offers instructors a number of options, we believe the way to use root questions is as follows:

1. An assignment consisting of several questions is assigned to the class. They are allowed to take the assignment as many times as they wish, and only their last score counts. They are thus incented to score 100% on the assignment. We recommend assignments of 4-5 questions. Too few questions allows them to guess randomly and eventually get a perfect score; too many questions creates too much risk that they will accidentally get something wrong and have to start over again.

2. Students open the assignment the first time. OTC gives them the questions in random order (to make it harder for students to share information about "question number 3"). The system also chooses at random one correct answer and three wrong answers, and presents those in random order. If the student can answer all the questions, that is fine. More likely, they will have trouble with one or more, so they do the best they can and submit their answers.

3. The student studies the material needed to answer the questions with which they are having trouble, off-line. For example, if they don't know how to compute the join, they could reread the material on that topic, attend office hours, or ideally, state what they don't understand in an email and get an instantaneous explanation from the global-TA system mentioned in Section 2.2. They then compute the entire join of the given relations, as if the question had been the long-answer problem "compute this join."

4. The student opens the assignment a second time, this time prepared with the answers to the general questions from which they will receive new, random instances. For example, if they have computed the join of R and S, then whichever tuple of that join appears among the four answers, they will have no trouble identifying it.

4. THE OTC TEAM

The OTC system is the work of Murti Valiveti and his team at Gautami Software, and of Ramana Yerneni and Alan Beck at Stanford. Root questions have been developed primarily by Ramana and me. Alan implemented the SQL lab and its questions.

As of the time of writing this document, mid-March, 2003, OTC has supported three course offerings. First was CS145 (Introduction to Databases) at Stanford in the Fall quarter of 2002. In the Winter quarter, we supported the second Stanford course, CS245, as well as a course at North Carolina State that used some questions from both of the Stanford courses. We are in the process of beginning support for several more courses for the Spring quarter. We appreciate the efforts of Qi Sun at Stanford and Rada Chirkova at NCSU to work with the system as it is being developed.