A Knowledge-Based Teaching System for SQL

Antonija Mitrovic
Computer Science Department, University of Canterbury
Private Bag 4800, Christchurch, New Zealand
tanja@cosc.canterbury.ac.nz

Abstract: The paper presents SQL-Tutor, an intelligent teaching system for SQL programming. SQL-Tutor is designed as a guided discovery learning environment and supports problem solving, conceptual and meta-learning. The system uses Constraint-Based Modelling to form models of its students. We present design issues by focusing on the system's architecture. Student modelling and the generation of pedagogical actions are discussed in the light of tailoring instructions towards a particular student.

1 Introduction

Structured Query Language (SQL) is the dominant database language today, containing data and view definition statements, as well as data manipulation statements. Although SQL is a simple and highly structured language, students have many difficulties learning it. This paper presents SQL-Tutor, an Intelligent Teaching System (ITS) that helps students in overcoming these difficulties. SQL-Tutor is designed as a practice environment; it supposes that students have previously been exposed to the concepts of database management in lectures. Therefore, the system is not a substitute for the conventional style of education, but a complement to it. The system currently covers only the SELECT statement of SQL, but the same approach could be used with other SQL statements. This focus on the SELECT statement does not reduce the importance of the system, because queries cause most misconceptions for students. Moreover, many of the concepts covered by SELECT are directly relevant to other SQL statements and other relational database languages in general.

The paper examines the problems of learning SQL firstly, and then presents the architecture of the system in [Overview of SQL-Tutor] and examines the system's components in the following three sections. The directions for future research and conclusions are reached in the last section.

2 Difficulties with Learning SQL

Students experience many problems when learning SQL. Some errors come from the burden of having to memorize database schemas; incorrect solutions may contain incorrect table or attribute names. Other errors come from misconceptions in the student's understanding of the elements of SQL and the relational data model in general. Some of the concepts students find particularly difficult to grasp are grouping and restricting grouping. Join conditions and the difference between aggregate and scalar functions are another two common sources of confusion. Other researchers report the same student misconceptions [Kearns et al. 1997].

SQL is usually taught in classrooms, by solving problems on the blackboard, complemented by lab exercises. However, students find that it is not easy to learn SQL directly by working with a RDBMS, because error messages are limited to the syntax only. [Fig. 1] illustrates a situation in which a student is required to specify a SELECT statement with five clauses, as shown in the correct solution. When the student enters an incorrect solution, typically the error message generated by a RDBMS (Ingres in this case) will not be of much help. The RDBMS can only complain about the syntactic error. The same figure illustrates the kind of messages[1] the

[1] Note that the student would usually be offered only one message at a time, as governed by the pedagogical rules. Here we show all possible messages for illustration.

student may obtain from the system. SQL-Tutor can generate messages about semantic errors as well; in this case, the student specified two tables in the FROM clause, when only one of them (MOVIE) is really needed.

Example 1: For each director, list the director's number and the total number of awards won by comedies he/she directed if that number is greater than 1.

Correct solution:

```
SELECT DIRECTOR,SUM(AAWON)
FROM MOVIE
WHERE TYPE='comedy'
GROUP BY DIRECTOR
HAVING SUM(AAWON)>1
```

Student's solution:

```
SELECT DIRECTOR,SUM(AAWON)
FROM DIRECTOR JOIN MOVIE
ON DIRECTOR=DIRECTOR.NUMBER
WHERE TYPE='comedy'
```

INGRES: E_USOB63 line 1, The columns in the SELECT clause must be contained in the GROUP BY clause.

SQL-Tutor:

- You do not need all the tables you specified in FROM!
- You need to specify the GROUP BY clause! The problem requires summary information.
- Specify the HAVING clause as well! Not all groups produced by the GROUP BY clause are relevant in this problem.
- If there are aggregate functions in the SELECT clause, and the GROUP BY clause is empty, then SELECT must consists of aggregate functions only.

Figure 1: Inadequacy of Feedback from a RDBMS

3 Overview of SQL-Tutor

SQL-Tutor is a knowledge-based system that supports students in learning SQL. As in the case with other ITSs, SQL-Tutor tailors instructional sessions to the needs, knowledge, learning abilities and general characteristics of its students. SQL-Tutor is based on guided discovery, one of the teaching styles commonly found in ITSs. Guided discovery is based on the idea that students should be given opportunities to discover things themselves, rather than being told about them. There are psychological studies [Anderson 1993] which show that students learn better from discovery than from direct instruction and that such knowledge is retained for longer than when learning by being told. Of course, unrestricted exploration is not advisable, especially for novices, as students may waste too much time wandering. The solution is found in providing guidance, in form of solicited or unsolicited hints from the system.

The system is designed as a problem-solving environment and as such is not intended to replace classroom instruction, but to complement it. We assume that students are already familiar with the database theory and fundamentals of SQL. Students work on their own as much as possible and the system intervenes when the student is stuck or asks for help. In such a way, students maintain a feeling of control.

SQL-Tutor is implemented in Allegro Common Lisp [Allegro 1996] on SUN workstations and PC compatibles (see [Mitrovic 1997] for more details). The components of the system (illustrated in [Fig. 2]) are the interface, a pedagogical module that determines the timing and content of pedagogical actions, and a student modeller (CBM), that analyzes student answers. There is no domain module, as usual in ITSs that can solve problems given to students. There are two reasons for not having a domain module. Firstly, database queries are given in a natural language; however, the current state-of-the-art in Natural Language Processing (NLP) is still far from being able of handling various problems present in such queries, such as references and synonyms. There is a possibility to avoid NLP: the text of the problem may be represented not in its natural-language form, but in a form that could be the product of NLP, as done in [Anderson et al. 1995]. However, it is hard not to build parts of the solution into such a representation. Furthermore, even if we overlook the NLP problem, the knowledge required to write SQL queries is very fuzzy and it would be very difficult to develop a problem solver in this area.

Nevertheless, an ITS must be able to evaluate student answers. SQL-Tutor does that by comparing student solutions to the correct ones. That is the reason for SQL-Tutor to require ideal solutions to problems. In order to be able to check the correctness of the student's solution, SQL-Tutor uses domain knowledge represented in the form of constraints, described in more detail in [Student Modeller].

The system contains definitions of several databases, implemented on the RDBMS used in the lab. New databases can easily be added to SQL-Tutor, by supplying the same SQL files used to create the database in the RDBMS. SQL-Tutor also contains a set of problems for specified databases and the ideal solutions to them.

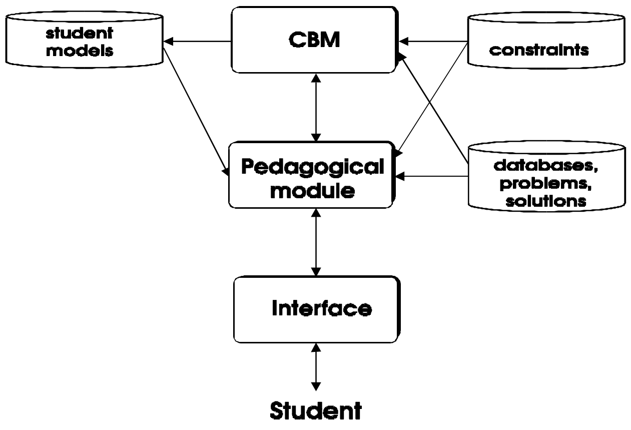


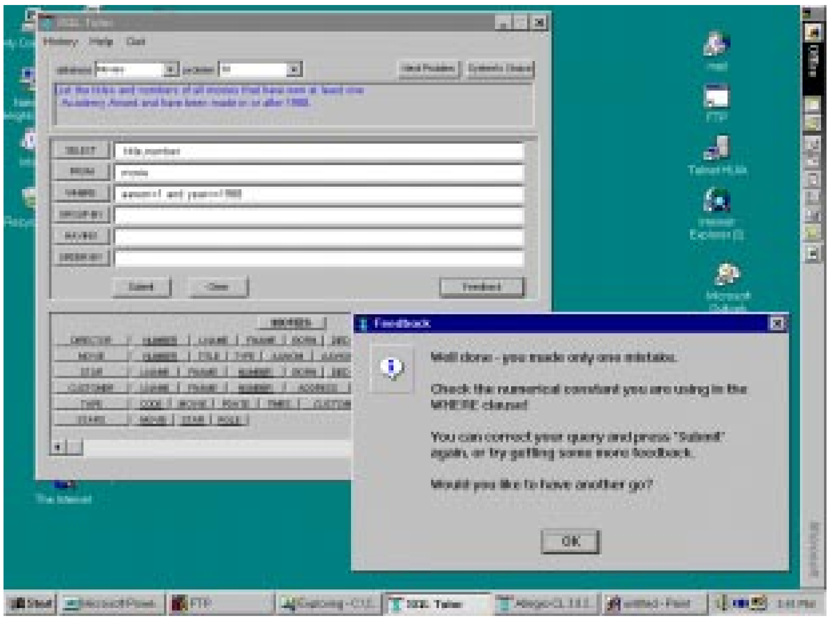
Figure 2: Architecture of SQL-Tutor

At the beginning of a session, SQL-Tutor selects a problem for the student to work on. When the student enters a solution, the pedagogical module (PM) sends it to the student modeller, which analyzes the solution, identifies mistakes (if there are any) and updates the student model appropriately. On the basis of the student model, PM generates an appropriate pedagogical action (i.e. feedback). When the current problem is solved, or the student requires a new problem to work on, PM selects an appropriate problem on the basis of the student model. The following subsections discuss the individual components in more detail.

4 Interface

The interface of SQL-Tutor, illustrated in [Fig. 3], has been designed with several pedagogical guidelines in mind. Generally, interfaces for ITSs should be robust, flexible, easy to use and understand. An interface is a mediating device; hence it must provide information about the system itself. At the same time, ITS interfaces are problem-solving environments and therefore should be similar to real environments, support the reification of goal structure and reduce the working-memory load of students

The interface of SQL-Tutor reduces the memory load by displaying the database schema and the text of a problem, by providing the basic structure of the query and also by providing explanations of the elements of SQL. The main window of SQL-Tutor is divided into three areas, which are always visible to the student. The upper part of the window displays the text of the problem being solved and the student can always remind him/herself easily of the elements requested in the query. The middle part contains the clauses of the SQL SELECT statement, thus visualizing the goal structure. Students need not remember the exact keywords used and the relative order of clauses. The lowest part displays the schema of the currently chosen database. The visualization of a schema is very important; all database users are painfully aware of the constant need to remember table and attribute names and the corresponding semantics as well. SQL-Tutor users can get the descriptions of databases, tables or attributes, as well as the descriptions of SQL constructs. The motivation here is to remove from the student some of the cognitive load required for checking the low-level syntax and to enable the student to focus on higher-level, query definition problems. SQL-Tutor supports the reification of the



List the titles and numbers of all movies that have won at least one Academy Award and have been made in or after 1988.

SELECT title,number
FROM movie
WHERE aawon>1 and year>=1988

Well done - you made only one mistake.

Check the numerical constant you are using in the WHERE clause!

You can correct your query and press "Submit" again, or try getting some more feedback.

Would you like to have another go?

Figure 3: The interface of SQL-Tutor

goal structure by visualizing the elements (clauses) of an SQL query. The student can obtain short descriptions of the roles of various clauses by selecting the appropriate clause or by asking for help from the main menu.

5 Student Modeller

A student modeller develops an understanding of the student's state of mind that can be used to generate instructional actions tailored to the particular student. The task of building a student model is extremely difficult and laborious, due to huge search spaces involved and the small amount of information to start from. Several researchers have pointed to the inherent intractability of the task [Holt et al. 1994, Ohlsson 1994, Self 1990]. If the goal is to model student's knowledge completely and precisely, student modelling is bound to be intractable. However, a student model can be useful although it is not complete and accurate [Ohlsson 1994, Self 1994, Stern et al. 1996]. Even simple and constrained modelling is sufficient for instruction purposes, and this claim is supported by findings that human teachers also use very loose models of their learners, and yet are highly effective in what they do [Holt et al. 1994, Self 1994]. SQL-Tutor uses Constraint-Based Modelling (CBM) [Ohlsson 1994] to form models of its students.

CBM reduces the complexity of student modelling by focusing on faults only. Domain knowledge is represented in the form of state constraints, where a constraint defines a set of equivalent problem states. An equivalence class triggers the same instructional action; hence the states in an equivalence class are pedagogically equivalent. The assumption here is that there can be no correct solution of a problem that traverses a problem state, which violates the fundamental ideas, or concepts of the domain. A violated constraint signals the error, which comes from incomplete and incorrect knowledge.

A state constraint is an ordered pair (*Cr*, *Cs*), where *Cr* is the relevance condition and *Cs* is the satisfaction condition. *Cr* is used to identify problem states, in which *Cr* is relevant, while *Cs* identifies the class of relevant states in which *Cs* is satisfied. Each constraint specifies the property of the domain, which is shared by all correct paths. In other words, if *Cr* is satisfied in a problem state, in order for that problem state to be a correct one, it must also satisfy *Cs*. Conditions may be any logical formulas, hence may consist of various tests on the problem state.

There are several advantages of CBM over other approaches. It does not require a runnable domain module and is computationally very simple, because student modelling is reduced to pattern matching. Conditions are combinations of patterns, and can therefore be represented in compiled forms, such as RETE networks [Forgy 1982]. Student modelling is very fast then: in the first step all relevance patterns are matched against the problem state. In the second step, the satisfaction components of constraints whose relevance conditions match the problem state are matched. If a satisfaction pattern matches the state, the constraint is satisfied, and the ITS is not to take any action; in the opposite case, the constraint is violated. The student model thus consists of all violated constraints. Furthermore, CBM does not require extensive studies of student's bugs, and is not sensitive to the radical strategy variability phenomenon. [Ohlsson 1994]. The approach is also neutral with respect to the pedagogy, since different pedagogical actions (immediate or delayed ones) may be generated on the basis of the model.

SQL-Tutor models students by looking at the student's solution and by comparing the student's solution to the ideal one. The constraint base of SQL-Tutor currently consists of 352 constraints, which are acquired by analyzing the domain knowledge [Elmasri & Navathe 1994, Pratt 1990] and from a comparative analysis of correct and incorrect student solutions. Each constraint has a unique number, and contains the relevance and satisfaction patterns. Additionally, there is a constraint description, and the name of the clause of the SELECT statement the constraint refers to. Relevance and satisfaction patterns can be any logical formulas, consisting of any number of conditions. Some conditions match parts of the student's solution to prespecified patterns or the ideal solution; other conditions are LISP functions.

It is well known that knowledge acquisition is a very slow, time-consuming and labour-intensive process. Anderson [1995] reports 10 or more hours necessary for induction of a production rule. When interviewing domain experts in order to acquire knowledge for expert systems, usually 2 to 5 production rules equivalents are identified per day. The time spent on identification, implementation and testing of SQL-Tutor's constraints averages at 1.3 hours per production, which is significantly shorter than times above. This may be the consequence of the same person serving as the domain expert and knowledge engineer (and the system developer, at that matter), but may also illustrate the appropriateness of the chosen formalism.

A student model in SQL-Tutor contains general information about the student, history of previously solved problems and information about the usage of constraints, as demonstrated in the solutions produced by the student. For each constraint, SQL-Tutor stores information about how many times it was found relevant for ideal solutions, how many times it was actually used by the student and how many times it was used correctly. This information is stored in terms of three indicators (*relevant*, *used* and *correct*), used by PM for selecting new problems as explained in [Pedagogical Module], and updated by the student modeller.

6 Pedagogical Module

Pedagogical module is the heart of the system; it selects problems to be given to students and generates appropriate instructional actions according to the student model. In SQL-Tutor, instruction can be individualized by generating feedback dynamically and selecting problems.

The level of feedback determines how much information is provided to the student. Currently, there are five levels of feedback in SQL-Tutor: positive/negative feedback, error flag, hint, partial solution and complete solution. At the lowest level (positive/negative feedback), the message simply informs the student whether the solution is correct or not and, in the later case, how many errors there are. An error flag message informs the student about the clause[2] in which the error occurred [Fig. 3]. A hint-type message gives more information about the type of error. Here, the student is given a general description of the error. This description is directly taken from the definition of constraint. Partial solution feedback displays the correct content of the clause in question, while the complete solution simply displays the correct solution of the current problem.

It was stated earlier that a student's solution may violate several constraints at the same time (as in [Fig. 1], where 5 constraints were violated simultaneously). In such cases, SQL-Tutor examines the violated constraints and selects one, which is likely to be a genuine misconception. That is, SQL-Tutor selects the constraint with the maximum difference between the *used* and *correct* indicators. The rationale here is the student has made the same error several times, and therefore instruction must start with that constraint. Currently, the student is told the total number of violated constraint, but the error messages only deal with one constraint at the time. This decision is based on our intuition that it would be much easier for the student to deal with errors one at a time.

Problems are also selected on the basis of a student model. SQL-Tutor examines the student model and selects a problem for constraints that the student is not sure about (i.e., the one with maximal *used-correct*). Another

[2] In case that there are several massages in various clauses, the pedagogical module will select one of them to start with.

possibility for the problem to be posed to the student is a problem that requires the use of a constraint that has not been used by the student so far.

SQL-Tutor also allows the student to select the problem on his/her own. Such an approach introduces randomness in the coverage of constraints, which can mean that the student in practising the use of some known constraint or even introducing new ones. Admittedly, such problem selection strategies are very simple and we are currently developing more sophisticated strategies.

7 Conclusions

SQL-Tutor is an intelligent teaching system, based on learning-by-doing. It supports three kinds of learning: conceptual, problem solving and meta-learning. The student can learn about concepts and elements of SQL. SQL-Tutor is a problem-solving environment which supports acquisition of domain knowledge in a declarative form (i.e. constraints) and strengthening of this knowledge in practice. The system provides assistance in problem solving and arguments against incorrect actions. Finally, SQL-Tutor encourages meta-learning by supporting self-explanation on the basis of error messages and correct solutions given to the student.

The system has been shown to a number of database teachers, and all were very supportive and expressed great enthusiasm for using it in their own courses. The first evaluation study is scheduled for early April 1998. We expect that a hands-on approach will encourage most students to experiment more fully and acquire a more in-depth understanding of SQL.

There are many possibilities for extending this research. The evaluation study will provide the data for completing the constraint base, and tuning the interface. A more sophisticated way of selecting new problems is the current focus of our research, as well as the development of a set of pedagogical rules that would govern the selection of appropriate amount of help for each student. We also plan to connect the system to a RDBMS, in order to allow students to view the results of their queries, once they are completed successfully. Porting the system to Web is another of planned tasks. There are many related areas in the database world, such as relational algebra and calculus, data modelling or normalization, which could serve as domains for other small instructional tools and be connected with SQL-Tutor into a “database exploration world”.

8 References

[Allegro 1996] *Allegro Common Lisp* (1996). Franz Inc.

[Anderson 1993] Anderson, J.R. (1993). *Rules of the Mind.* Hillsdale, NJ: Lawrence Erlbaum Associates.

[Anderson et al. 1995] Anderson, J.R, Corbett, A.T., Koedinger, K.R., Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *Journal of the Learning Sciences,* 4(2), 167-207.

[Elmasri & Navathe 1994] Elmasri, R., & Navathe, S.B. (1994). *Fundamentals of Database Systems*. Redwood: Benjamin/Cummings.

[Forgy 1982] Forgy, C.L. (1982). Rete: a Fast Algorithm for the Many Pattern/Many Object Pattern Matching Problem. *AI*, 19, 17-37.

[Holt et al. 1994] Holt, P., Dubs, S., Jones, M., Greer, J.E. (1994). The State of Student Modelling. *Student Modeling: the Key to Individualized Knowledge-based Instruction.* Berlin: Springer-Verlag, 3-35.

[Kearns et al. 1997] Kearns, R., Shead, S., Fekete, A. (1997). A Teaching System for SQL. *Australasian Computer Science Education Conference*. ACM Press, 224-231.

[Mitrovic 1997] Mitrovic, A. (1997). SQL-Tutor: a Preliminary Report. *Technical Report TR-COSC 08//97.* Computer Science Department, University of Canterbury.

[Ohlsson 1994] Ohlsson, S. (1994). Constraint-based Student Modeling. *Student Modeling: the Key to Individualized Knowledge--based Instruction.* Berlin: Springer-Verlag, 167-189.

[Pratt 1990] Pratt, P.J. (1990). *A Guide to SQL.* Boston: Boyd & Fraser.

[Self 1990] Self, J. A. (1990). Bypassing the Intractable Problem of Student Modeling. *Intelligent Tutoring Systems: at the Crossroads of Artificial Intelligence and Education.* Norwood: Ablex, 107-123.

[Self 1994] Self, J. (1994). Formal Approaches to Student Modeling. *Student Modeling: the Key to Individualized Knowledge--based Instruction.* Berlin: Springer-Verlag, 295-352.

[Stern et al. 1996] Stern, M., Beck, J., Woolf, B.P. (1996). Adaptation of Problem Presentation and Feedback in an Intelligent Mathematics Tutor. *Intelligent Tutoring Systems,* New York: Springer-Verlag, 603-613.

Acknowledgements

The work presented here was supported by the University of Canterbury research grant U6242. We thank Kendrick Elsey and Andrew King for implementing the interface.