An Intelligent SQL Tutor on the Web

Antonija Mitrovic
tanja@cosc.canterbury.ac.nz

Kurt Hausler
kjh63@student.canterbury.ac.nz

Intelligent Computer Tutors Group
Computer Science Department, University of Canterbury
Private Bag 4800, Christchurch, New Zealand

Abstract: We present **SQLT-Web**, a Web-enabled intelligent teaching system for the SQL database language. The system observes students' actions and adapts to their knowledge and learning abilities. Constraint-Based Modelling is used to model students. We describe the system's architecture in comparison to architectures of other existing Web-enabled tutors. All tutoring functions are performed on the server side, and we explain how **SQLT-Web** deals with multiple students. An initial evaluation of **SQLT-Web** has been done in a database course in May 1999. The students have enjoyed the system's adaptability and found it a valuable asset to their learning.

1 Introduction

Intelligent Teaching Systems (ITS) offer the advantage of individualized instruction without the expense of one-to-one human tutoring. Although numerous ITSs have been developed to date, they are mostly used in research environments, and only a few have been used by large numbers of students in real classrooms. The main cause of such limited use of existing systems is the complexity of ITS development, and the difficulties with providing robust and flexible systems. The area is young; there are no well established methodologies or development tools. The time needed for the development of one hour of instruction in an ITS is estimated to 100 hours of development time. Furthermore, the hardware platforms available in most schools are not the ones developers prefer, and porting systems between platforms is in no way a straightforward task. Fortunately, Web-enabled versions of ITSs have the potential to reach a much wider audience as they face significantly fewer problems with hardware and software requirements.

We have developed **SQL-Tutor**, a standalone system for teaching SQL (Structured Query Language) [Mitrovic 1998a, 1998b]. The system has been used by senior computer science students at the University of Canterbury and has been found easy to use, effective and enjoyable [Mitrovic and Ohlsson, 1999]. The system has been developed in Allegro Common Lisp [Allegro 1998] and is available on MS Windows and Solaris. Besides local users, in only three months, more than four hundred people worldwide have downloaded the Windows version of the system[1]. However, we wanted to open the system to a wider audience, and avoid problems with porting between various platforms. In this paper we present **SQLT-Web**, a Web-enabled version of **SQL-Tutor**. We discuss the advantages and disadvantages of commonly used architectures for Web-based systems first, followed by a discussion of the architecture we adopted for **SQLT-Web.** Then, we describe the features of the system that support students' learning and discuss how multiple students are handled

[1] **SQL-Tutor** is available for downloading from http://www.cosc.canterbury.ac.nz/~tanja/ictg.html

simultaneously. We present our initial experiences with the system in section 4, and further research directions in the final section.

2 Architectures of Web-enabled Tutors

Web-enabled tutors offer several advantages in comparison to standalone systems. They minimize the problems of distributing software to users and hardware/software compatibility. New releases of tutors are immediately available to everyone. More importantly, students are not constrained to use specific machines in their schools, and can access Web-enabled tutors from any location and at any time.

Several architectures for Web-enabled tutors have emerged so far, all based on the client-server architecture. If we consider the location at which the tutoring functions are performed, these architectures may be classified into the following three categories:

- The ***centralized architecture***, illustrated in Figure 1, consists of a Web server, an application server and student interface. The application server and the Web server run on the server side, while the student interface is displayed in a Web browser on the client's machine. The application server performs all tutoring functions. The interface consists of a set of HTML pages. The student interacts with HTML entry forms, and the information is sent to the Web server, which passes the student's requests and actions to the application server.

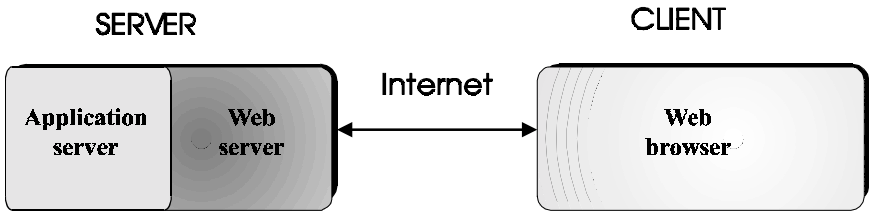

Figure 1. Centralized architecture

There are several mechanisms for communication between the server and the interface, the most common of which is the use of CGI (Common Gateway Interface) programs. Information sent by a Web browser is processed by an external CGI program, and the results are sent back in the form of new HTML pages. Examples of systems that follow this philosophy are **WITS**, a symbolic equation-solving tutor [Okazaki et al. 1996] and **PAT-Online**, an algebra tutor [Ritter 1997].

Another option is using programmable Web servers, which can be extended with the application code, thus eliminating the need for external CGI programs. This is the architecture that **SQLT-Web** is based upon, discussed in more detail in Section 3.2. Other examples include **ELM-ART**, a Lisp tutor [Brusilovsky et al. 1996] and **AST**, a statistics tutor [Specht et al. 1997].

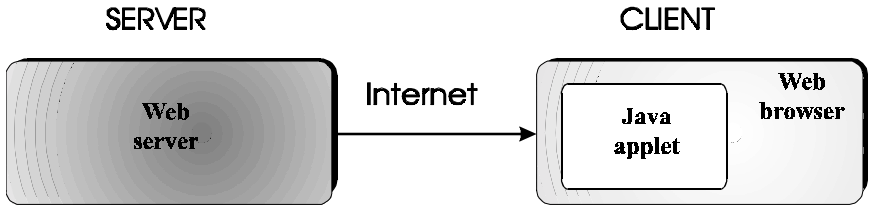

Figure 2. Replicated architecture

- In the ***replicated architecture*** (Figure 2), the entire tutor resides in a Java applet that needs to be downloaded and is executed on the student's machine. All tutoring functions are therefore

performed on the client's machine, while the server is only used as a repository of software to be downloaded. An example is a tutor [Vassileva 1997] developed in the **DCG** authoring tool. The tutor is a downloadable Java applet, consisting of a conceptual structure of the domain, a planner, a student model and the executor of generated plans. Its interface is delivered via HTML pages, with attached Java applets that carry out interaction with the student and diagnosis of their answers. The same philosophy underlies a trauma care tutor [Johnson 1998], where a copy of a pedagogical agent, named **ADELE**, is run on each student's computer, and performs all tutoring actions. The central server manages course materials and performs administrative functions.

- In the ***distributed architecture*** *(*Figure 3) tutoring functionality is distributed between the client and the server. The exact policy on distributing the functions may vary. Most often, the application server consists of a student modeler, which creates and maintains student models for all users, a domain module, capable of solving and/or selecting problems, and a pedagogical module. The user interface is usually Java-based and may perform some teaching functions. Additional functionality in the interface includes immediate feedback for each problem solving step, and interactive graphics and simulations.

  Communication between the interface and the application server does not necessarily involve the Web server; it is possible to establish a direct TCP connection between the applet and the application server in order to speed up the system. **AlgeBrain**, **Medtec** and **Belvedere** are based on such an architecture. **AlgeBrain** [Applet et al. 1999] supports students while learning to solve algebraic equations. A downloadable Java applet provides an engaging user interface involving an agent that reacts to a student's action, and provides immediate feedback on each student's step. **Medtec** is a Web-based anatomy tutor [Eliot 1997], the application server of which is developed in the CL-HTTP server. Java applets are used to provide interactive graphics. **Belvedere** [Suthers and Jones 1997] is a system for learning scientific inquiry skills. Java is used to deliver the user interface, while the application server is written in a variety of tools.

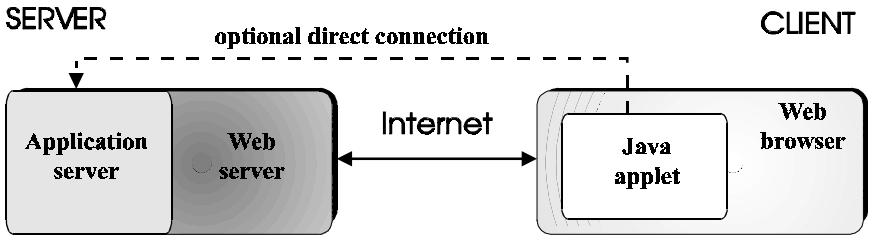

Figure 3. Distributed architecture

In all architectures, the student needs a Web browser, which is a common requirement today. The differences between various browsers are small, and typically there are not many problems in ensuring that a Web-enabled system can be used via various browsers.

The amount of effort involved in building a tutor with a replicated architecture is the same as building a standalone system. These systems are very fast, as all processing is done on the client's machine. However, a significant limitation of this architecture is the fact that the student model is stored on the machine where the tutor has been executed. Therefore, the student always needs to use the system from the same machine if he/she wants to benefit from the summaries of previous sessions stored in the student model, otherwise the knowledge about previous sessions would be lost and the system would not be able to adapt to the student easily. One interesting solution to this problem may be found in [Vassileva 1997] and **ADELE**, where copies of student models are also kept on the server between sessions for persistent storage. Although this solution removes the requirement that a student always has to use the tutor from the same machine, there is still a problem if a network error occurs before the

student completes a session, as the most recent information about student's performance will then be lost.

In both replicated and distributed architectures, it is necessary for a student to download software in order to start using a system – a task that some students may find frustrating. Furthermore, it is necessary to download each new release of a tutor to benefit from the improvements. In the case of a centralized architecture, there are no such problems.

A significant advantage of the centralized and distributed architectures is the fact that all student models are kept in one place (on the server) and the student can use the system from any machine. Additional knowledge structures, needed by the expert or pedagogical module, may be shared. A problem with these two architectures may be the reduced speed, caused by communications between the client and the server. The situation might be better for a system with distributed architecture, as some of the tutoring actions are performed on the client side and hence the number of communications is reduced. However, communicating between the interface and the server in a distributed architecture may require special techniques, which introduces additional complexity to system development.

3 The development of SQLT-Web

The starting point for the development of **SQLT-Web** was **SQL-Tutor**, a standalone system for teaching SQL. Here we briefly describe the standalone version and then explain the process of converting it into a Web-enabled tutor.

3.1 The standalone version

Figure 4 illustrates the architecture of **SQL-Tutor**. For a detailed discussion of the system, see [Mitrovic 1998] or [Mitrovic and Ohlsson 1999]; here we present only some of its most vital features. **SQL-Tutor** consists of an interface, a pedagogical module which determines the timing and content of pedagogical actions, and a student modeller (CBM), which analyzes student answers. The system contains definitions of several databases, implemented on a DBMS, and a set of problems and the ideal solutions to them. **SQL-Tutor** contains no domain module. In order to check the correctness of the student's solution, **SQL-Tutor** compares it to the correct solution (specified by a teacher), using domain knowledge represented in the form of constraints. It uses Constraint-Based Modeling [Ohlsson 1994] to model knowledge of its students.

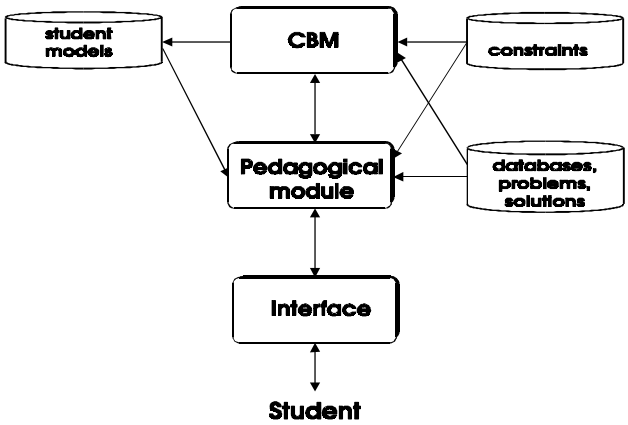

Figure 4: Architecture of SQL-Tutor

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 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, the pedagogical module generates an appropriate pedagogical action (i.e. feedback). When the current problem is solved, or the student requires a new problem to work on, the pedagogical module selects an appropriate problem on the basis of the student model.

3.2 The architecture of SQLT-Web

Starting from the standalone system, we have developed a list of requirements for a Web-enabled tutor. We wanted to maintain a centralized repository of student models and support multiple simultaneous students, thus giving students freedom to access the system at any time and any place. We also wanted to eliminate the need to download software, and therefore decided to use the centralized architecture, which fulfils all requirements. An integrated Web development environment embodied by the Common Lisp Hypermedia Server[2] (CL-HTTP) [Mallery 1994] was selected for implementing the system. We preferred this option to using CGI because of the disadvantages of the latter; when CGI is used to process user requirements, it is necessary to run a separate CGI program in response to each web request. In order to maintain consistency between various requests in a single session, it is necessary to implement a student model in an external database instead of maintaining knowledge structures in the memory. This mechanism would be too complex in a research environment, characterized by frequent changes in requirements, and therefore we decided to use CL-HTTP server, which eliminates the need for CGI.

CL-HTTP is a fully featured HTTP server developed in Common Lisp. Since the original **SQL-Tutor** was also implemented in Common Lisp, CL-HTTP appears to be an optimal platform. CL-HTTP supports application development by directly extending the server using Common Lisp programming. Developers may define Lisp functions to handle incoming requests, and generate HTML pages as responses. CL-HTTP is based on multi-threaded programming, and creates a separate thread to respond to each client. As several students who use the system concurrently share some components of SQLT-Web, it is necessary to introduce a locking mechanism to ensure non-interference between various sessions. The system also needs to maintain multiple student models and to associate every request to the student model of the corresponding student. We discuss how SQLT-Web supports multiple students in Section 3.4.

Figure 5 presents the architecture of **SQLT-Web**, which is the extension of the architecture of the standalone system. We have re-implemented the interface, introduced a session manager and extended the domain knowledge structures. At the beginning of an interaction, a student is required to enter his/her name, which is necessary in order to establish a session. The session manager records all student actions and the corresponding feedback in a log. It also requires the student modeler to retrieve the model for the student, if there is one, or to create a new model for a student who interacts with the system for the first time.

Each action a student performs in the interface is first sent to the session manager, as it has to link it to the appropriate session. Then, the action is sent to the pedagogical module, which decides how to respond to it. If the submitted action is a solution to the current problem, the pedagogical module sends it to the student modeler, which diagnoses the solution, updates the student model, and sends the result of the diagnosis back to the pedagogical module. The pedagogical module then generates feedback. If the student has requested a new problem, the pedagogical module consults the student model in order to identify the knowledge elements the student has problems with, and selects one of

[2] CL-HTTP server is available from http://www.ai.mit.edu/projects/iiip/doc/cl-http/home-page.html

the predefined problems that feature identified misconceptions. Students may also ask for additional explanations, which are dealt with by the pedagogical module.

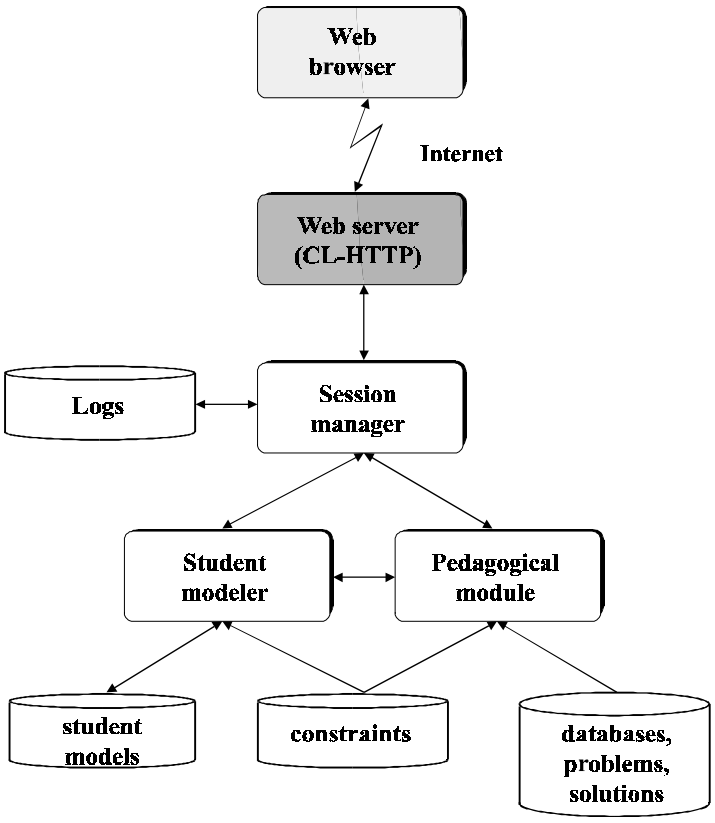

Figure 5. The architecture of SQLT-Web

3.3 Interface

The interface of **SQLT-Web**, illustrated in Figure 6, has been designed to be robust, flexible, and easy to use and understand. It 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 page is divided into three areas. The upper part displays the text of the problem being solved and students can remind themselves easily of the elements requested in queries. 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. Schema visualization 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. Students 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.

When a solution is submitted, the pedagogical module generates feedback on it, offers the possibilities of working on the same problem (if there were mistakes in the student's solution), logging off, or going on to the next problem, which may be selected by the student or the system.

3.4 Supporting multiple students

SQLT-Web maintains information about a student in his/her student model, which summarizes student's knowledge and the history of the current and previous sessions. Initially, **SQLT-Web** acquires information about a student through a login screen. Individual student models are stored permanently on the server, and retrieved for each student's session. Students who are inactive for a long period of time are automatically logged off (after 120 minutes) and their models are moved back to long term storage.

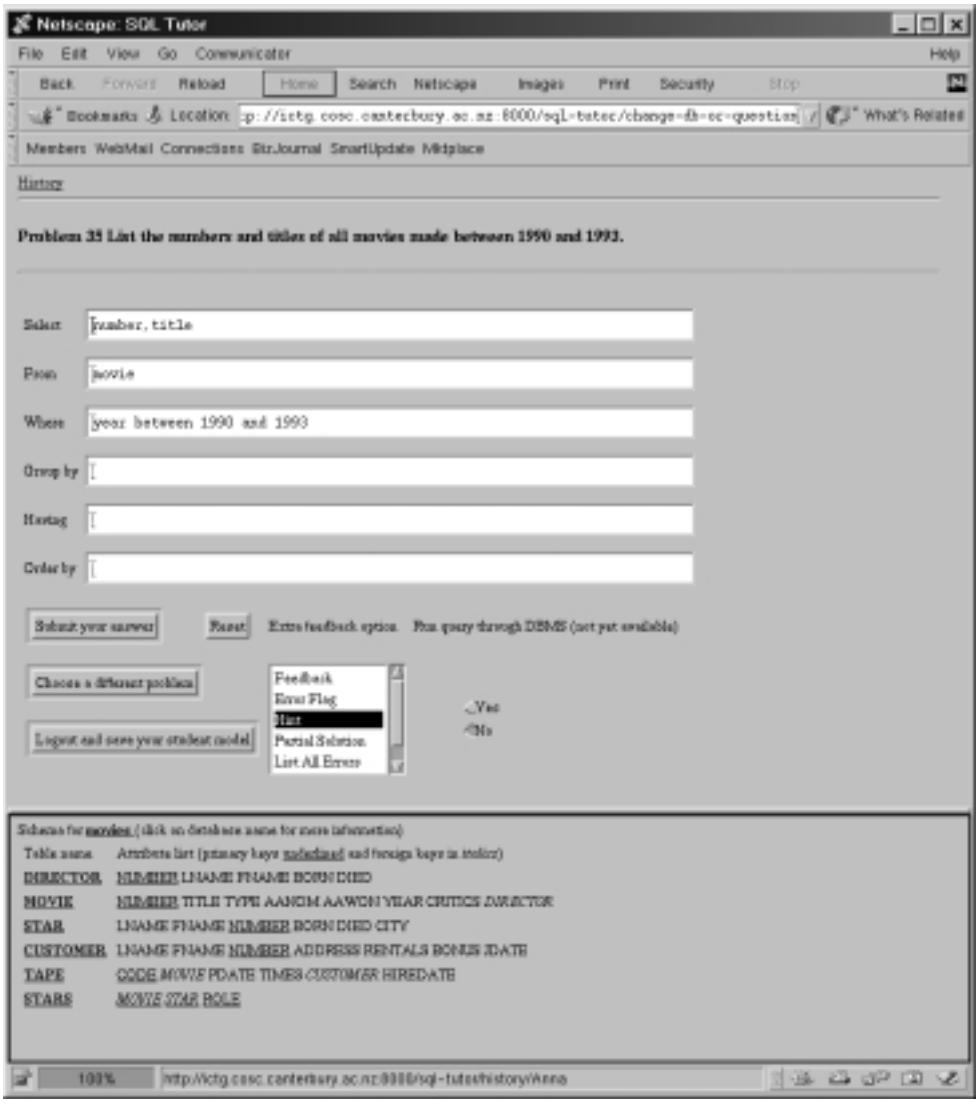

Figure 6. The interface of SQLT-Web

A web-based tutor with a central repository of student models must respond to requests of individual students. The system must be able to associate each request to the appropriate student model. Some Web-enabled systems use cookies or IP numbers to identify the student who made a request. Those two approaches were not suitable in our case. It was not possible to use the IP number, as several students might be using the same machine. Also, we did not want to use cookies for identification purposes because various browsers deal with them in different ways. Instead, we identify students by their login name, which is embedded in a hidden tag of HTML forms and sent back to the server. If a

student accesses a page by specifying the URL instead of accessing it through a form, then user name is appended to the end of the URL.

It is also necessary to store student-specific data separately from data about other students. All processing is carried out within a single address space, and therefore there must be a uniform mechanism for identifying students and associating requests to corresponding student models. In order to achieve this, we use a hash table that maps the string representing a student name to their student object, which contains all details pertaining to the students, such as a timestamp for automated logout, the history of the current session, the cache of the previous incorrect attempt, the feedback buffer, currently selected database and problem, etc.

Student modeler uses constraint networks [Mitrovic 1998a,b] to diagnose a student's solution. There may be many students submitting their solutions to the system concurrently, and therefore these knowledge structures must be locked while processing a single student's solution. Whenever a student submits a solution, the system needs to check whether the constraint networks are available (i.e., to make sure that the processing of a previous solution has been completed and the locks on the networks have been released) before the current solution can be processed.

4 Initial evaluation

SQLT-Web has been used in a two-hour lab session with 33 senior computer science students in May 1999. The students had learnt about SQL in 6 lectures and had at least eight hours of hands-on experience of query definition prior to using the system. All students' actions were recorded and the students filled out a questionnaire at the end of the session. We have evaluated **SQLT-Web** on two dimensions: usability and learning.

Questions	Agree	Disagree
1. One hour with SQLT-Web is more valuable than one hour of lectures/labs	36	51
2. Would you like to use SQLT-Web more?	84	3
3. Would you recommend SQLT-Web to other students?	84	0
4. Do you find the display of the schema understandable?	93	3

Table 1. Responses (in percentages) from the user questionnaire

The responses to the user questionnaire revealed that students enjoyed learning with the system and appreciated its adaptive features. The majority of students (77%) reported that they needed less than 10 minutes to start using the system, 9% needed 30 minutes; finally, two students reported spending most of the two hours becoming familiar with the system. The students enjoyed the system (questions 2, 3 and 6 in Tables 1 and 2[3]). Consistent with these findings, we observed that the students continued to use the system on their own after the study. The students found the interface easy to use (question 7), and appreciated having the schema of the currently selected database (question 4, Table 1).

The user questionnaire contained several questions about learning. When asked to rate how much they learned from working with the system, the average rating was 3.1 (question 5). There was no agreement on whether one hour with SQLT-Web was more valuable than one hour of lectures/labs (question 1). One explanation for the relatively low values on these variables is that many of the students had already encountered the relevant databases and problems in their prior laboratory exercises, and therefore found no unseen problems in **SQLT-Web**. This is not a deficiency of the system as it is easy to add new databases and problems. The average answer for the helpfulness of

[3] The percentages given in the table do not add up to 100%, as not all students answered all questions.

feedback messages was 2.9 (question 8). There were a few suggestions on how to provide additional useful information, such as connecting the system to a DBMS, so that queries can actually be run and results inspected.

Questions	1	2	3	4	5
5. How much did you learn about SQL from using the system?	6	15	36	30	6
6. Did you enjoy learning with SQL-Tutor?	0	9	27	42	21
7. Do you find the interface easy to use?	0	6	24	54	15
8. Do you find help messages useful?	9	24	33	24	6

Table 2. Responses to questions based on the Likert scale (1 - not at all, 5 - very much)

The majority of students appreciated the exploratory, hands-on approach, learning at their own pace and found learning with **SQLT-Web** to be more personal than lectures. Other students commented that human input was still necessary sometimes. Some of the suggestions soon to be implemented included requests for more examples of how to generate queries and more SQL-specific help.

We have also analyzed student logs in order to see what kind of learning is taking place. Since we represent knowledge in the domain of SQL in terms of constraints, we looked at how students acquire them and apply them. In earlier work [Mitrovic and Ohlsson 1999], the evaluation of SQL-Tutor showed that constraints represented psychologically appropriate units of knowledge, and learning followed a smooth curve when plotted in terms of constraints. We have performed the same analysis in SQLT-Web. Figure 7 shows the decrease in the number of constraints failed, as a function of the number of times each constraint was relevant. Students learn constraints independently of each other, and the degree of mastery of a given constraint is a function of the amount of practice on that unit. We plan to perform a wider evaluation study in October 1999 within the University of Canterbury and then open **SQLT-Web** for outside students.

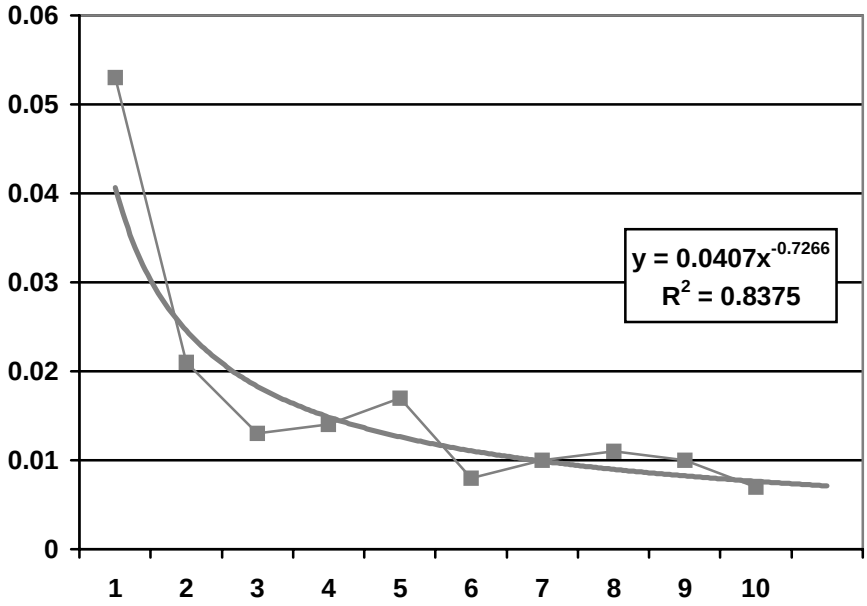

Figure 7. Probability of violating a constraint as a function of number of occasions when that constraint was relevant, averaged over subjects

5 Conclusions

The Web has introduced a new paradigm for building widely accessible intelligent teaching systems. A very important aspect of Web-based tutors is the ability to use sophisticated tools for knowledge-intensive components of systems, and develop the interfaces in platform-independent ways.

SQLT-Web is a Web-enabled system for teaching SQL. The system is an extension of a standalone system developed in Common Lisp, and we re-used its code for the Web-based extension. **SQLT-Web** is developed in the CL-HTTP server. It is based on a centralized architecture, where all tutoring functions are performed on the server, and the only functions performed on the client's side are the user interaction ones. The amount of data that needs to be transferred between the client and the server in **SQLT-Web** is small due to the nature of the domain, and therefore the centralized architecture is feasible.

SQLT-Web has been used by senior computer science students in May 1999, and has been found to be effective and easy to use. The majority of students appreciated the exploratory, hands-on approach, learning at their own pace and found learning with **SQLT-Web** to be more personal than lectures. We plan to improve some interface features and introduce new ways of selecting problems based on curriculum. One of the current projects involves the development of an animated pedagogical agent, which will be used in the next evaluation study scheduled for October 1999. In the long term, we plan to introduce support for self-explanation, and allow students to engage in more profound types of learning.

6 References

Allegro Common Lisp (1998). Franz Inc.

Alpert, S., Singley, M., Fairweather, P. (1999) Deploying Intelligent Tutors on the Web: an Architecture and an Example. *Int. J. Artificial Intelligence in Education*, 10, 183-197.

Brusilovsky, P., Schwarz, E., Weber, G. (1996) ELM-ART: an Intelligent Tutoring System on Wolrd Wide Web. In C. Frasson, G. Gutier, A. Lesgold (eds), *Intelligent Tutoring Systems, ITS'96*, 261-269.

Eliot, C. (1997) Implementing Web-Based Intelligent Tutors, *Proc. Workshop on Adaptive Systems and User Modeling on the World Wide Web*, UM-97, 37-42.

Johnson, W.L., Shaw, E., Ganeshan, R. (1998) Pedagogical Agents on the Web. *Proceedings of a workshop on "Intelligent Educational Systems on the Web, ITS'98.*

Mallery, J.C. (1994) A Common LISP Hypermedia Server. *Proc 1st Int. Conf. On the World Wide Web*.

Mitrovic, A. (1998a). Learning SQL with a Computerized Tutor, Proc. 29th SIGCSE Technical Symposium, 307-311.

Mitrovic, A. (1998b). A Knowledge-Based Teaching System for SQL, *Proc. ED-MEDIA'98*, T. Ottmann, I. Tomek (eds.), 1027-1032.

Mitrovic, A., Ohlsson, S. (1999). Evaluation of a constraint-based tutor for a database language, *Int. J. Artificial Intelligence in Education*, 10, 3-4, to appear.

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

Okazaki, Y., Watanabe, K., Kondo, H. (1996). An Implementation of an Intelligent Tutoring System on the World-Wide Web: Individualizing Tutoring Mechanism in the WWW Framework. *Educational Technology Research*, 19(1), 35-44.

Ritter, S. (1997). PAT-Online: a Model-Tracing Tutor on the World-Wide Web. P. Brusilovsky, K. Nakabayashi, S. Ritter (eds) *Proceedings of workshop on Intelligent Educational Systems on the World Wide Web,AI-ED'97, 11-17.*

Specht, M., Weber, G., Heitmeyer, S., Schoch, V. (1997) AST: Adaptive WWW-Coursware for Statistics. *Proc. Workshop on Adaptive Systems and User Modeling on the World Wide Web*, UM-97, *91-96.*

Suthers, D., Jones, D. (1997) An Architecture for Intelligent Collaborative Educational Systems. B.de Boulay, R. Mizoguchi (eds) *Proc. AI-ED'97*, 55-62.

Vassileva, J. (1997) Dynamic Course Generation on the WWW. *Proceedings of AIED'97.*

Acknowledgements

The work presented here was supported by the University of Canterbury research grant U6242. We thank Brent Martin for helping with data analysis.