An Intelligent SQL Tutor on the Web

Antonija Mitrovic

International Journal of Artificial Intelligence in Education 13 (2003) 171–195
IOS Press

An Intelligent SQL Tutor on the Web

Antonija Mitrovic, *Intelligent Computer Tutoring Group, Computer Science Department*
University of Canterbury, Private Bag 4800, *Christchurch, New Zealand*
tanja@cosc.canterbury.ac.nz
http://www.cosc.canterbury.ac.nz/~tanja

Abstract. The paper presents **SQLT-Web**, a Web-enabled intelligent tutoring system for the SQL database language. **SQLT-Web** is a Web-enabled version of an earlier, standalone ITS. In this paper we describe how the components of the standalone system were reused to develop the Web-enabled system. The system observes students' actions and adapts to their knowledge and learning abilities. We describe the system's architecture in comparison to the 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. The system has been open to outside users since March 2000. **SQLT-Web** has been evaluated in the context of genuine teaching activities. We present the results of three evaluation studies with the University of Canterbury students taking database courses, which show that **SQLT-Web** is an effective system. The students have found the system a valuable asset to their learning.

Keywords. Web-based ITS, architectures for adaptive and intelligent Web-based educational systems, intelligent problem solving support via the Web, student modelling and student model servers in the Web context, empirical studies of Web-based adaptive and intelligent educational systems.

INTRODUCTION

Intelligent Tutoring 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. Despite the fact the area is not young, there are no well-established methodologies or development tools. 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). 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 & Ohlsson, 1999). The system has been developed in Allegro Common Lisp (Allegro, 1998) and is available on MS Windows and Solaris. Besides local users, two thousand people worldwide

have downloaded the Windows version of the system[1] in 20 months starting in May 1999. However, we wanted to open the system to a wider audience, and avoid problems with porting between various platforms. The goal of this paper is to present **SQLT-Web**, a Web-enabled version of SQL-Tutor that was developed by reusing the standalone version, and to show that the developed system is effective. We discuss the advantages and disadvantages of commonly used architectures for Web-based educational 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 simultaneously. We present our experiences with the system in section 4, and further research directions in the final section.

ARCHITECTURES OF WEB-ENABLED ADAPTIVE EDUCATIONAL SYSTEMS

Web-enabled educational systems offer several advantages in comparison to standalone systems. They minimize the problems of distributing software to users and hardware/software compatibility. New releases of systems 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 ITSs have emerged so far, all based on the client-server architecture. If we consider the location at which the teaching functions are performed, three types of architectures can be identified: centralized, replicated and distributed. In all systems that will be used to illustrate the three architectures, the student needs a Web browser, which is a common requirement today. Although one of the promises of Web is platform-independence, the differences between various browsers are not negligible, and often require substantial effort to ensure that a Web-enabled system can be used via any browser.

In this section we present only the general features of the three architectures. More detailed surveys of various approaches and technologies used to build Web-enabled educational software can be found in (Brusilovsky, 1999; Alpert et al., 1999; Eliot, 1997; Stern et al., 1997).

Centralized architecture

In the ***centralized architecture***, illustrated in Figure 1, 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 entry forms. Information entered by the student is sent to the Web server, which passes the student's requests and actions to the application server.

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

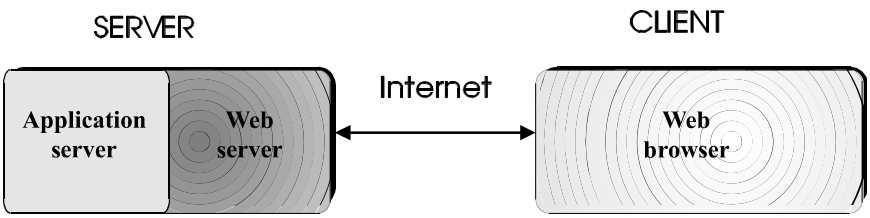


Fig. 1. Centralized architecture

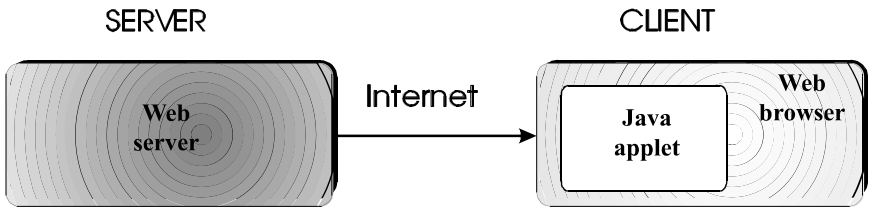
The most common mechanism for communication between the server and the interface is CGI (Common Gateway Interface). For each particular URL the server is to respond to, there is an external CGI program, which processes the data and generates the results in the form of a new HTML page. Examples of systems that follow this philosophy are **WITS**, a symbolic equation-solving tutor (Okazaki, Watanabe & Kondo, 1996), **PAT-Online**, an algebra tutor (Ritter, 1997), and **ID** (Siekmann et al., 2000), a multimedia system for teaching mathematics. The problem with using CGI is the necessity 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 be able to relate each incoming request to a particular student.

Another option is using programmable Web servers, which can be extended with the application code. Common Lisp Hypermedia Server[2] (CL-HTTP) (Mallery, 1994) is an example of such programmable Web servers. CL-HTTP is a fully featured HTTP server developed in Common Lisp, which supports application development by directly extending the server using Common Lisp programming. CL-HTTP uses CGI and enables developers to define Lisp functions to handle these incoming requests, and generate HTML pages as responses. To generate responses, developers can use a special set of HTML-generating functions. This is the architecture that **ELM-ART**, a Lisp tutor (Brusilovsky, Schwarz & Weber, 1996; Weber & Brusilovsky, 2001), and **AST**, a statistics tutor (Specht, Weber, Heitmeyer, & Schoch, 1997) are based upon. The development of **SQLT-Web** largely follows the approach used in these two systems, and is further discussed in the next section.

Replicated architecture

In the ***replicated architecture*** (Figure 2), the entire tutor resides in a Java client 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 ADIS (Warendorf & Tan, 1997), a Java-based Web-enabled ITS for teaching data structures. ADIS is available as a Java applet that runs in a Web browser, or a Java application, which includes its own graphical user interface.

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



Fig, 2. Replicated architecture

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.

Distributed architecture

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. Vassileva (1997) describes the **DCG** authoring tool. The server contains teaching materials, concept structures, which describe the pedagogical structure of various domains, and the planner. The student requests a course by specifying a domain and a learning goal, and the planner develops a course for the student based on his/her student model. The course is then downloaded to the client's site, together with the Executor, a Java application that controls the execution of the course. The 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, Shaw, & Ganeshan, 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 this kind of replicated architecture, the amount of communication between the application server and the client is small. The client needs be to downloaded at the beginning of the interaction, but after that all the processing is done on the client. At the end of the session, the student model is posted to the application server, where all the student models are kept between sessions.

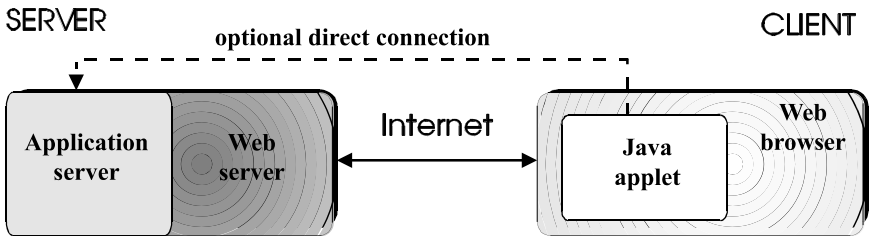


Fig. 3. Distributed architecture

Another common architecture for distributed tutors involves the application server consisting 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** (Alpert, Singley, & Fairweather, 1999, 2000) 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 & 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. **German Tutor** (Heift & Nicholson, 2001) is an adaptive multimedia system, where the user interface is implemented as a Java applet. **Virtual Campus PROLOG Tutor** (Peylo, Thelen, Rollingen & Gust, 2000) is an ITS that provides a problem-solving environment for students. The user interface is implemented as a Java client. The system does limited student diagnosis, but also offers a possibility of human interference.

Discussion

As previously discussed, replicated Web-enabled systems are fast, as all the processing is done on the client site, but are limited by the fact the student model is stored on the local machine. 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.

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 centralized architecture is appropriate in situations when the level of interaction is low and when there is no direct manipulation of objects on the screen requiring immediate response from the system. In domains that require highly interactive interfaces, replicated and distributed architectures would be more appropriate. 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.

THE DEVELOPMENT OF SQLT-WEB

The starting point for the development of **SQLT-Web** was **SQL-Tutor**, a standalone system for teaching SQL. The functionality of **SQLT-Web** is identical to that of the stand-alone version. In this section we firstly describe the standalone version and then explain the process of converting it into a Web-enabled tutor. Then we describe the various components of the system, starting with the interface and the knowledge base. Brusilovsky (1999) identifies three core ITS technologies: curriculum sequencing, intelligent analysis of student's solutions and interactive problem solving support. Here we discuss the first two of these. The third technology, interactive problem solving support, provides the student with help on each step while solving a problem, and is not supported in **SQLT-Web**.

The standalone version

Figure 4 illustrates the architecture of **SQL-Tutor**. For a detailed discussion of the system, see (Mitrovic, 1998a; Mitrovic & Ohlsson, 1999); here we present only some of its 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; Mitrovic, 1998b) to model knowledge of its students. The constraint base is described later in a separate section.

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.

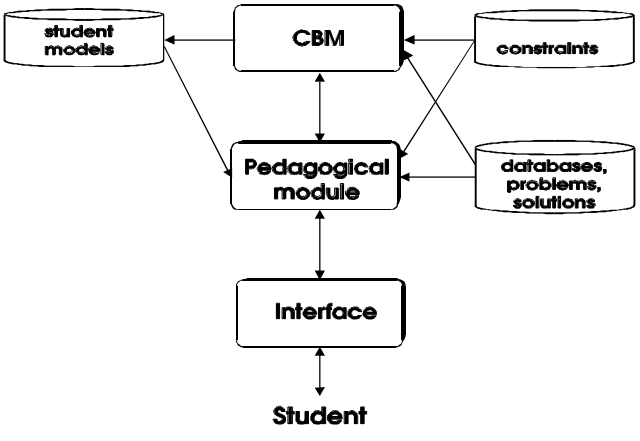


Fig. 4. Architecture of SQL-Tutor

The architecture of SQLT-Web

Starting from the standalone system, we have developed a list of requirements for a Web-enabled tutor. Firstly, we wanted to reuse as much of the existing system as possible. Another requirement was 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 from any place. Since the amount of information that needs to be processed is small (only the student's solution and selections are of interest), we decided to use the centralized architecture, which fulfils all these requirements. An integrated Web development environment embodied by CL-HTTP was selected for implementing the system. We preferred this option to using CGI directly because CL-HTTP supports application development by extending the server using Common Lisp programming. Since the original **SQL-Tutor** was also implemented in Common Lisp, CL-HTTP appears to be an optimal platform. 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 a later section.

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 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 modeller 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 student is also assigned a level. At the beginning of the first session with the system, the student selects the appropriate initial level her/himself, from three possibilities: "novice", "intermediate", or "experienced". This level is later updated in accordance with observations of the student's behavior: it is incremented if he/she solves two or more problems consecutively at or above his/her current level, within three attempts each. Both problem and student levels are used for problem selection, as described in the section on curriculum sequencing.

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

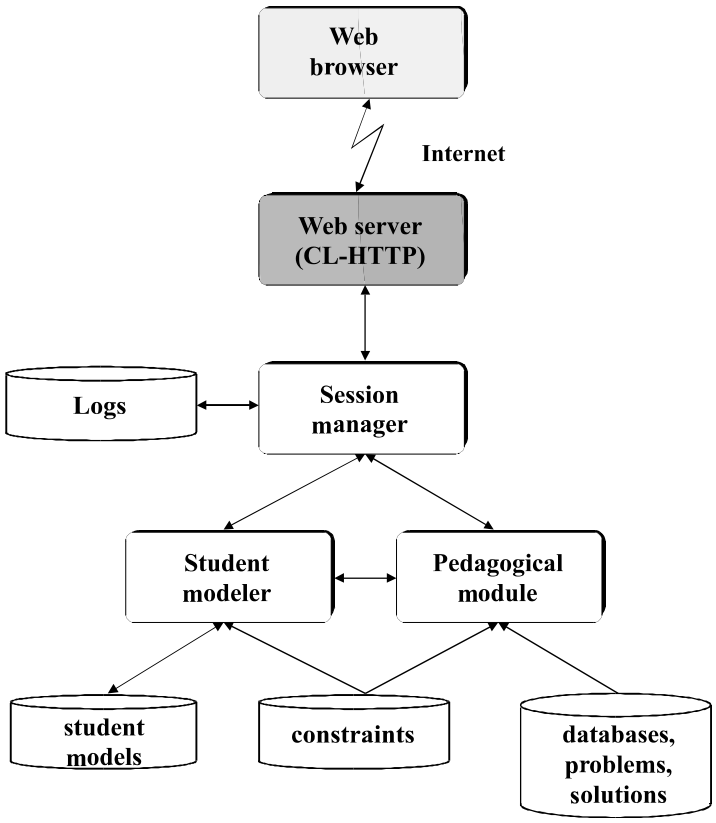


Fig. 5. The architecture of SQLT-Web

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 four 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 left 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 middle right part is where the feedback from the system is presented. 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. The student is also able to view the history of the session, ask for a query to be run on the database and specify the kind of feedback required.

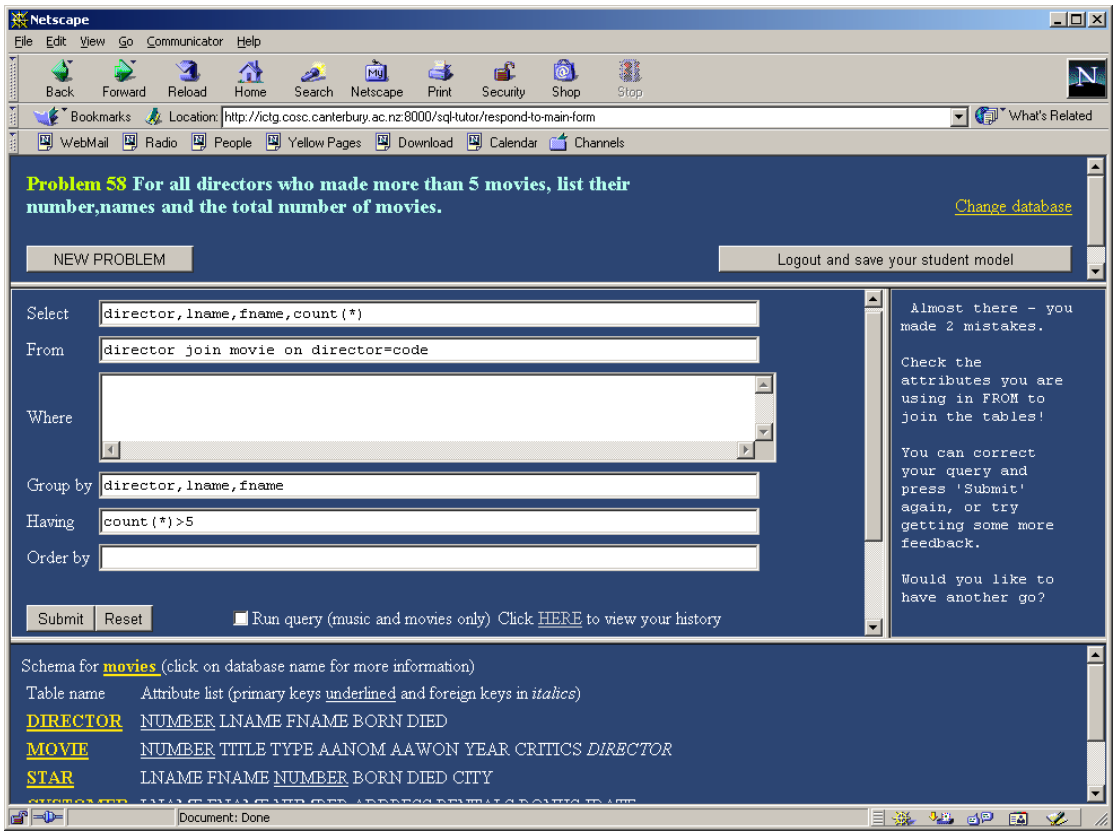


Fig. 6. Interface of the Web-enabled version of SQL-Tutor

Knowledge base

The knowledge about the domain that **SQLT-Web** contains is represented as a set of constraints. Constraint-Based Modeling (CBM) is a student modelling approach proposed by Ohlsson (1994), as a way of overcoming the intractable nature of student modelling. CBM arises from Ohlsson's theory of learning from errors (1996), which proposes that we often make mistakes when performing a task, even when we have been taught the correct way to do it. According to this theory, we make mistakes because the declarative knowledge we have learned has not been internalized in our procedural knowledge, and so the number of decisions we must make while performing the procedure is sufficiently large that we make mistakes. By practicing the task,

however, and catching ourselves (or being caught by a mentor) making mistakes, we modify our procedure to incorporate the appropriate rule that we have violated. Over time, we internalize all of the declarative knowledge about the task, and so the number of mistakes we make is reduced. Ohlsson describes the process of learning from errors as consisting of two phases: *error recognition* and *error correction*. A student needs declarative knowledge in order to detect an error. Only then can the error be corrected so that the solution used is applicable only in situations in which it is appropriate.

CBM starts from the observation that all correct solutions to a problem are similar in that they do not violate any of the basic principles of the domain. CBM is not interested in the exact sequence of states in the problem space the student has traversed, but in what state they are currently in. As long as the student never reaches a state that is known to be wrong, they are free to perform whatever actions they please. Constraints define equivalence classes of problem states. An equivalence class triggers the same instructional action; hence all states in an equivalence class are pedagogically equivalent. It is therefore possible to attach feedback messages directly to constraints. A violated constraint signals an error, which translates to incomplete/incorrect knowledge. The domain model is therefore a collection of state descriptions of the form:

> *"If <relevance condition> is true, then <satisfaction condition> had better also be true, otherwise something has gone wrong."*

In other words, if the student solution falls into the state defined by the relevance condition, it must also be in the state defined by the satisfaction condition in order to be correct.

SQLT-Web currently contains more than 600 constraints, and this number is likely to increase as new problems requiring new situations are added to the system. All constraints are problem-independent; they describe the basic principles of the domain, and do not involve any elements of problems directly. The constraints are modular, and are not related to each other. In order to identify constraints, we studied material presented in textbooks, such as (Elmasri & Navathe, 1999), and also used our own experience in teaching SQL. The constraints specify the syntax and semantics of SQL. Figure 7 illustrates three constraints, which are related to specifying the join condition in FROM. More examples can be found in (Mitrovic 1998b; Mitrovic & Ohlsson, 1999; Mitrovic et al., 2001).

The first part of each constraint is a unique number, followed by the hint message that will be displayed if the constraint is violated. The first constraint (110) checks the syntax. It is relevant when the JOIN keyword is used in FROM, and it is satisfied if the ON keyword is also used in the same clause. In other words, to specify a join condition in FROM, the student has to use both keywords.

Constraint 358 also deals with the join condition specified in FROM, but its relevance condition is more specific. In cases when the student has used both the JOIN and the ON keyword in FROM, the satisfaction condition checks whether the keywords and other elements (variables that stand for table and attribute names) have been specified in the right order.

Constraint 358 checks only the syntax of the FROM clause, while constraint 11 also checks the semantics. Its relevance condition specifies that the constraint is important for those solutions containing ON, JOIN and the corresponding variables in FROM, where two variables correspond to tables in the currently selected database, and one of the remaining two variables is an attribute of the first table. The satisfaction condition asserts that in such cases, the solution is correct if the

other attribute specified in the FROM clause is the attribute of the second table, and the two attributes are of the same type. Figure 6 illustrates a situation when this constraint is violated.

```
(p 110
"You need the ON keyword in FROM!"
(member "JOIN" (from-clause ss) :test 'equal) ;Relevance condition
;ss is the student's solution
(member "ON" (from-clause ss) :test 'equal) ;Satisfaction condition
"FROM")

(p 358
"Check the syntax for the JOIN and ON keywords in FROM!"
; Relevance condition
(and (member "JOIN" (from-clause ss) :test 'equalp)
     (member "ON" (from-clause ss) :test 'equalp))
; Satisfaction condition
(match '(?*d1 ?t1 ??s1 "JOIN" ?t2 ??s2 "ON" ?a1 "=" ?a2 ?*d2)
     (from-clause ss) bindings)
"FROM")

(p 11
"If the JOIN keyword is used in the FROM clause, the same clause should contain a join condition
specified on a pair of attributes from corresponding tables being joined."
; Relevance condition
(and (match '(?*d1 ?t1 ??s1 "JOIN" ?t2 ??s2 "ON" ?a1 "=" ?a2 ?*d2) (from-clause ss) bindings)
; FROM contains variable t1, JOIN followed by variable t2, the ON keyword and a comparison
     (valid-table (find-schema (current-database *student*)) ?t1)
; t1 is a table from the current database
     (valid-table (find-schema (current-database *student*)) ?t2)
; t2 is a table from the current database
     (attribute-of (find-table ?t1 (current-database *student*)) ?a1))
; a1 is an attribute of table t1
; Satisfaction condition
(and (attribute-of (find-table ?t2 (current-database *student*)) ?a2)
; a1 is an attribute of table t2
     (equalp (find-type ?a1) (find-type ?a2)))
; a1 and a2 must be attributes of the same type
"FROM")
```

Fig. 7. Three constraints

The last clause of each constraint identifies which part of the solution the constraint is dealing with (the FROM clause in these three constraints). As can be seen, the relevance and

satisfaction conditions are LISP clauses. They may contain any LISP predicate, but the most frequent predicate is *match*, which performs pattern matching.

The three illustrated constraints may be relevant at the same time, if the student has specified the join condition in FROM using the correct syntax (as in Figure 6). However, if there are syntax errors, only some of them would be relevant. For example, if the FROM clause of the student solution is *FROM MOVIE JOIN DIRECTOR,* the first two constraints will be relevant, while constraint 11 will not be relevant. Constraint 110 will be satisfied, while constraint 358 will be violated, thus generating a hint for the student.

As mentioned earlier, the system is not capable of solving the problems on its own, as there is no problem solver. Instead, there is an ideal solution for each problem. The system analyses the student's solution by matching it to constraints and ideal solutions. **SQLT-Web** has constraints that make sure that the student's solution contains all the necessary elements of the solution. Figure 8 shows two constraints of this type.

Constraint 207 is relevant when the join condition is specified in FROM in the ideal solution, and the student has used more than one table in FROM, and has not specified the join condition in WHERE. In such a case, the student solution will be correct if there is a join condition in FROM. Notice that this constraint only requires the JOIN keyword to appear in FROM, and does not care about its syntax. Constraints 110, 358 and 11 will make sure that the syntax is correct.

It is common in SQL to have two or more correct solutions for a problem, especially if the problem is complex. **SQLT-Web** contains only one correct solution to the problem. However, the system is capable of recognizing alternative correct solutions, as there are constraints that check for equivalent constructs in the student's and ideal solutions. Constraint 387 illustrates such capabilities of the system. A join condition may be specified in the FROM clause (using the JOIN and ON keywords), or in the WHERE clause, by specifying and equality comparison on the joining attributes. Constraint 387 is relevant if the student has a syntactically valid join condition in the FROM clause, and the ideal solution does not have the join condition in FROM. However, the ideal solution has a join condition specified in the WHERE clause, using two attributes that come from the same tables that the student has used. The constraint will be satisfied if the student uses the same attributes that are used as joining attributes in the ideal solution.

We believe that CBM is neutral with respect to the domain. Within the ICTG group, we have developed constraint-based tutors in domains very different from SQL. CAPIT is a system that teaches the rules of punctuation and capitalization in English (Mayo & Mitrovic, 2001). KERMIT teaches conceptual database design using the ER data model (Suraweera & Mitrovic, 2002), and NORMIT (Mitrovic, 2002) teaches data normalization, a procedural task. We have experienced no problems expressing the knowledge in these domains in terms of constraints. CBM is applicable both to procedural and declarative domains. Currently, we are considering other domains with different characteristics, to further test the generality of CBM. Creating constraint-based tutors tends to require less time and effort than cognitive tutoring. For a discussion of CBM in comparison to cognitive tutoring, please see (Mitrovic, Koedinger & Martin, 2003).

```
(p 207
"You need to specify the join condition in FROM!"

; Relevance condition
(and (null (slot-value is 'where))
; The WHERE clause of the ideal solution (is) is empty.
      (member "JOIN" (from-clause is) :test 'equalp)
; The ideal solution has a join condition in FROM.
      (> (length (find-names ss 'from-clause)) 1)
; There is more than one table in the FROM clause in the student's solution.
      (null (slot-value ss 'where)))
; The WHERE clause in the student's solution is empty.

; Satisfaction condition
(member "JOIN" (from-clause ss) :test 'equalp)
; The student should specify the JOIN condition in FROM.

"FROM")

(p 387

"Check the attributes you are using in FROM to join the tables!"

; Relevance condition
(and (match '(?*d1 ?t1 ??s1 "JOIN" ?t2 ??s2 "ON" ?a1 "=" ?a2 ?*d2) (from-clause ss) bindings)
      (valid-table (find-schema (current-database *student*)) ?t1)
      (valid-table (find-schema (current-database *student*)) ?t2)
      (attribute-of (find-table ?t1 (current-database *student*)) ?a2)
; There is a valid join condition in the student's FROM clause
      (not (member "JOIN" (from-clause is) :test 'equalp))
; The ideal solution does not contain a join condition in FROM.
      (member ?t1 (from-clause is) :test 'equalp)
      (member ?t2 (from-clause is) :test 'equalp)
; The same two tables that the student used appear in the ideal solution.
      (bind-all ?n1 (names (where is)) bindings)
      (attribute-of (find-table ?t1 (current-database *student*)) ?n1)
      (match '(?*d3 (?is ?n2 attribute-p) "=" ?n1 ?*d4) (where is) bindings)
      (attribute-of (find-table ?t2 (current-database *student*)) ?n2))
; There is a join condition in the WHERE clause in ideal solution

; Satisfaction condition
(and (same-attributes ?a1 ?n2) (same-attributes ?a2 ?n1))

"FROM")
```

Fig. 8. Two constraints that match the student's and ideal solutions

Intelligent analysis of student's solutions

SQLT-Web analyses the student's solution once when it is submitted, by matching it to the constraints and the ideal solution. A very important feature of CBM is its computational simplicity. CBM does not involve complex reasoning, as required by other student modelling

approaches. Instead, CBM reduces student modelling to pattern matching. Although pattern matching is simple, it can potentially be time-consuming, especially in situations when the number of patterns is large. However, patterns can be represented in compiled forms, such as RETE networks (Forgy, 1982), which are very fast.

In order to speed up the matching process, all constraints in **SQLT-Web** are compiled into two networks resembling RETE networks. One network contains the relevance conditions of all constraints, while the other one contains the satisfaction conditions. The main idea is the same as in RETE: reuse as much of the previous work as possible. Each network contains three types of nodes: input, test and output nodes. Input nodes are entry points for the network. Each input node contains a test that appears as the initial test in the relevance condition of a constraint. If there are several constraints with the same initial test, they will share the same input node. The difference between the networks used in **SQLT-Web** and RETE networks is that a test node has just one input, so the structures are trees, not unrestricted networks. Each test node contains a test to be applied on the student's and possibly ideal solution. If several constraints share the same test, but the subsequent tests are different, the test node will be connected to as many other test nodes as there are different subsequent tests. A constraint that has m tests in its relevance condition will result in a path of length $m+1$ in the relevance network, consisting of an input node, $m-1$ test nodes and an output node. The input node and the initial test nodes may be shared with other constraints. The last node in the path is the output node, and it contains only the constraint number.

When the student submits a solution to be checked, it is propagated through the relevance network first. Each input or test node is evaluated by applying the test it contains on the student's solution. If a test is satisfied, the list of bindings will be propagated together with the student's solution to test nodes connected to the current one. If the test is violated, then the particular path within the network will be abandoned, and the connecting nodes will not be activated. The propagation process continues until the output nodes are reached. In the case of the relevance network, the output nodes correspond to constraints that are relevant to the student's solution. If an output node has been reached in the satisfaction network, the corresponding constraint is satisfied.

The analysis of a student's solution is done in two steps in **SQLT-Web**. In the first step, all relevance patterns are matched against the problem state. In the second step, the student's solution is propagated through the satisfaction network, but only for those constraints that were found relevant in the first step. The number of relevant constraints per problem ranges from 78 for the simplest problems, to more than two hundred for complex ones.

Ohlsson (1994) views the student model as consisting of the relevant and violated solution. This is actually the short-term model of the student, which reflects student's performance on the current task. However, for an ITS to function properly, it is also necessary to represent student's long-term model. A student model in **SQLT-Web** contains information about general student characteristics (name, level of expertise, history etc) and the model of the student's knowledge. The latter is represented as an overlay upon the constraint base. For each constraint, **SQLT-Web** stores the history of its usage, and the percentage of correct use. The constraint history simply specifies for each occasion of application whether the student has used the constraint correctly or incorrectly. The percentage of correct use is calculated on the basis of n most recent attempts, thus giving more weight to the recent past and allowing for learning/forgetting. We have also experimented with a probabilistic user model, a discussion of which is outside of the scope of

this paper. The interested reader is referred to (Mitrovic et al., 2001) for a discussion of the use of Bayesian networks in **SQLT-Web**.

Curriculum sequencing

Curriculum sequencing is a set of planning techniques used in educational systems to provide the student with the most appropriate sequence of elementary knowledge units to learn or problems to solve. Brusilovsky (1999) identifies two kinds of curriculum sequencing in ITSs and other educational systems: active and passive. *Active sequencing* is characterized by the existence of a learning goal, expressed in terms of one or more domain concepts to be learned. *Passive sequencing* does not require a learning goal, but simply reacts to the current student's action by offering suitable learning material or a next problem to work on.

Only passive sequencing is implemented in **SQLT-Web**. The system contains a set of constraints that specify the basic principles of SQL. At the moment, there is no curriculum structure of the domain. For each database, there is a list of problems ordered in accordance with their complexity. The student may go over these problems in turn or may ask the system to select the appropriate problem. In the latter case, the system selects the best problem for a student on the basis of student model, according to the following procedure. When the student requests a new problem, the system calculates the percentage of correct use for each constraint, and then identifies the *focu*s constraint. This is a constraint with the lowest percentage of correct use. The system then identifies a problem that is relevant to the focus constraint from the pool of unsolved problems whose level is within +1 or –1 of the student's current level. We have also experimented with the probabilistic student model, using Bayesian networks to predict the performance of a student on a problem. This prediction was used as a criterion to select a problem. For details of this experiment, please see (Mitrovic et al., 2001).

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.

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. We did not want to use cookies for identification purposes because various browsers deal with them in different ways. Furthermore, cookies reside on a single/specific machine and would introduce the same limitations as the replicated architecture, preventing the student from using the system from different machines. 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 following

a link instead of accessing it through a form, then the user's 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.

As explained earlier, the student modeler uses the relevance and satisfaction networks 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 these 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. The propagation of a student's solution through these networks is extremely fast, and we have not experienced any delays in the evaluations performed due to the locking mechanisms. As discussed in the following section, the maximum number of students participating in the studies was 70, so the effect of having a much larger number of concurrent students needs to be investigated.

EVALUATION

The stand-alone version of the system was evaluated in 1998 (Mitrovic & Ohlsson 1999), showing that the system had a significant effect on students' knowledge after a single, 2-hour long session. Here we report on three evaluation studies performed on **SQLT-Web**, in May 1999, October 1999 and late 2000, referred hereon as studies 1, 2 and 3 respectively. General information about the studies is given in Table 1. All studies were carried out at the University of Canterbury, with Computer Science students enrolled in database courses.

Table 1
Details of the evaluation studies

Study	Timing	Students	Length	Purpose of study
1	May 1999	33	2 hours	Feedback evaluation
2	October 1999	34	2 hours	Pedag. agent; Probabilistic student model
3	Sep-Oct 2000	70	7 weeks	Meta-cognitive skills

In the first two studies, the students used **SQLT-Web** in a single, two-hour session, during their normal lab time. Prior to using the system, the students had all attended six lectures about SQL, and completed at least eight hours of hands-on experience of query definition. All students' actions were recorded, and the students filled out a questionnaire at the end of the session. There were several observers present at each evaluation, who reported that the students were quite interested in interacting with the system and exploring its various functions.

Study 1 involved all senior-year students enrolled in a database course (33 students). The goal of this study was to evaluate the effectiveness of various types of feedback provided by the system. Students were randomly allocated to one of two versions of the system: the first gave restricted feedback, while the second generated all levels of feedback. In this paper, we report only on the evaluation of the system in general; for the details of the evaluation of feedback, please see (Mitrovic & Martin, 2000).

Study 2 was performed in October 1999, and involved all second-year students taking an introductory database course. In addition to the questionnaire, the students sat a pre- and post-test. Three versions of the system were used in the study: the basic version, a version which generated probabilistic student models and used them to select problems, and a version in which feedback was presented via an animated pedagogical agent. Students were randomly assigned to one of the versions. The evaluation of the various versions of the system is irrelevant to this paper, but we refer the interested reader to (Mitrovic & Suraweera, 2000) for the details of the evaluation of the pedagogical agent, and to (Mayo & Mitrovic, 2000) for the details of the evaluation of the probabilistic student model and the appropriateness of problems selected on the basis of this model.

The third study was longer. The system was demonstrated in a lecture at the beginning of September 2000. The course involved a test on SQL a month and a half after the system was introduced. The experiment was set up this way so that the students may use the system over several weeks. Out of 142 students enrolled in the course, 79 have used the system. The total interaction time varied a lot, because there were students who had only briefly looked at the system. We excluded the logs of nine students who had not attempted any problems. The goal of study 3 was to analyze students' metacognitive skills, and the results are described in (Mitrovic, 2001).

As mentioned above, each of these studies had a specific focus. In this paper, we report of the general findings, and evaluate **SQLT-Web** on two dimensions: usability and learning. The results are given in the following two subsections.

Subjective evaluation

This section presents a summary of the students' answers to the user questionnaire in studies 1 and 2. The purpose of the questionnaire given to students at the end of the session was to evaluate the students' perception of **SQLT-Web**. The questionnaire, which is included in Appendix A, consisted of 16 questions, most of which were based on the Likert scale with five responses ranging from *very much* (5) to *not at all* (1). Students were also able to give free-form responses.

The students who participated in study 1 reported having more experience with SQL outside

Table 2

Responses (in percentages) from the user questionnaire

Responses (study1/study2)	Agree	Disagree
Would you recommend SQLT-Web to other students?	84/94	3/0
Do you find the display of the schema understandable?	93/88	3/0

the university (45% of the group) than the students who participated in study 2 (23%). The responses to the user questionnaire revealed that students enjoyed learning with the system and appreciated its adaptive features. The majority of students (77% in study 1, and 85% in study 2) reported that they needed less than 10 minutes to start using the system. 9% of students in both studies reported that they needed 30 minutes to learn about the system's features. Finally, two students reported spending most of the two hours becoming familiar with the system. The students enjoyed the system (Tables 2 and 3[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 (Table 3). Database schemas were deemed understandable (Table 2), and there were many free-form responses to this question, which show that the students appreciated having the database schema.

The user questionnaire contained several questions about learning. When asked to rate how much they learned from working with the system (Table 3), the average ratings were 3.1 (study 1) and 4.1 (study 2). There are three reasons for this rating to be higher in study 2. Firstly, the majority of students in study 1 had already encountered the relevant databases and problems in their prior laboratory exercises, and therefore found no unseen problems in **SQLT-Web**. A new database was added to the system in time for study 2, which may have had challenged the students more. Furthermore, the students who participated in study 1 were senior-level students, while in the second study we worked with the second year students, who needed more help and the additional practice and feedback were more helpful to this group. Thirdly, the students from study 1 reported more experience with using SQL outside of the class (45%), while only 23% of students from study 2 reported such experience. Therefore a significant portion of the students in study 1 had little to learn, which is in contrast to the situation in study 2. The same reasons explain why the rating for the usefulness of feedback (question 6) was higher in study 2 (4.2) than in study 1 (2.9).

Table 3
Responses (in percentages) to questions from the questionnaire

Responses (study 1/study 2)	1 (not at all)	2	3	4	5 (very much)
How much did you learn about SQL from the system?	6/0	15/6	36/20	30/50	6/23
Did you enjoy learning with the system?	0/0	9/9	27/23	42/29	21/38
Do you find the interface easy to use?	0/0	6/6	24/26	54/41	15/23
Do you find feedback useful?	9/0	24/9	33/12	24/26	6/47

From the written comments, it can be seen that 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 at times.

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

Learning with SQLT-Web

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 & 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 9 shows the decrease in the number of violated constraints, as a function of the number of times each constraint was relevant. The degree of mastery of a given constraint is a function of the amount of practice on that unit. There is not much difference between the three student populations, as the graphs for all three evaluation studies are close to each other. In other words, the students tend to acquire constraints at pretty similar paces.

In study 3, students were free to work with SQLT-Web whenever they wanted to, and the session lengths were not constrained. The average number of sessions that the students had with the system was 2. The length of sessions also varied greatly: the shortest session was only one minute long, while the longest took 300 minutes. The average duration of a session in the experiment was 47.45 minutes, and the average total interaction time was 95.6 minutes. The average number of problems attempted in a single session was 6.65, while the average number of solved problems per session was 1.5. The total number of solved problems per student (during the total interaction time) ranged from 1 to 44, with the average being 10.26 (67.5%).

Study 3 students also sat a pre- and a post-test. The pre-test was administered at the beginning of a student's first interaction with **SQLT-Web**. The post-test was administered on paper to all students enrolled in the course 7 weeks after the start of the evaluation study. Both tests consisted of three multi-choice questions of similar complexity. The maximal number of marks for both tests was 7.

Table 4 summarizes the results of the pre- and post-tests. The experimental group consists of 70 students who tried at least one problem, and the control group consists of the rest of the class. Because the pre-test was administered when the students logged on to the system for the first time, we only have the results of the pre-test for the experimental group. Therefore, it is not possible for us to compare the pre-test results for the control and the experimental groups.

These two groups listened to exactly the same number of lectures and labs, and sat the same post-test. The difference is that the students in the control group have not used **SQLT-Web**. The results of the experimental group on the post-test are higher than the results of the control group, and the difference is significant ($t=2.79$, $p<0.005$). However, this result is not irrefutable, as the experiment was not controlled. The experimental group consisted of volunteers, who are usually more motivated students.

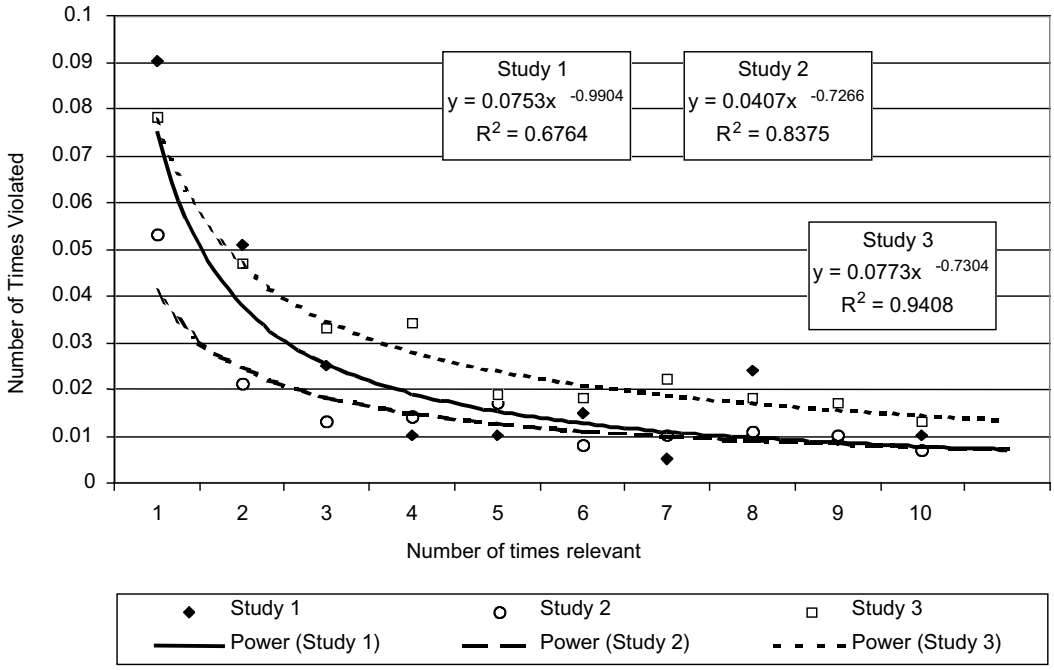


Fig. 9. Mastery of constraints

Table 4
Pre- and post-test results for study 3

Group	Student	Pre-test mean	Pre-test SD	Post-test mean	Post-test SD
Experimental	70	4.02	1.52	5.01	1.24
Control	62			4.3	1.6

We also analysed the performance of students in the experimental group in relation to the time they have spent with the system. Of all the students in the experimental group, we identified only those who have solved at least one problem, and divided the students into two groups. Group A consists of 35 students who have spent less time than the average total time, while group B contains those student who interacted with the system longer. Group A students have improved their mean mark in the post-test more than students from group B, whose marks in the pre-test were better. However, the difference in the improvements is not significant. This is so because the time spent with the system is not correlated strongly to the number of problems solved. In one extreme case, a student managed to solve only one problem after spending 300 minutes with the system.

Table 5
The results on the tests in relation to the total interaction time

Group	Students	Time (min)	Problems solved	Pre-test	Post-test
A	35	41.88 (27.42)	7.54 (6.78)	3.97 (1.40)	5.08 (1.38)
B	21	223.85 (107.19)	20.05 (13.58)	4.19 (1.53)	5.05 (1.02)

Since we are not able to differentiate idle time from thinking time, we also analysed the performance of students in relation to the number of problems they were able to solve. Table 6 presents the results of this analysis. For this analysis, we post-hoc split the students into group 1, consisting of students who solved less problems than the mean number of solved problems, and group 2 containing the remaining students. Students in group 2 scored higher on both the pre- and the post-test, and the difference in improvements is significant (t=1.06, p<.15, _=.16). Therefore, student who solve more problems also obtain higher scores in the post-test.

Table 6
The results on the tests in relation to the number of solved problems

Group	Students	Problems solved	Time (min)	Pre-test	Post-test
1	34	4.65 (3.50)	64.23 (69.30)	3.65 (1.35)	4.88 (1.36)
2	22	24.81 (10.18)	181.05 (129.11)	4.63 (1.39)	5.36 (0.95)

CONCLUSIONS

The Web has introduced a new paradigm for building widely accessible intelligent educational systems. A very important aspect of Web-based tutors is the ability to develop the interfaces in platform-independent ways. This paper presented **SQLT-Web**, 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. The system runs on a dedicated machine, and we have not experienced any problems with network delays.

SQLT-Web has been used in database courses at the University of Canterbury by senior computer science students since 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. The system has also been used by outside users, since March 2000. We have already started work on introducing support for self-assessment and other meta-cognitive strategies, in order to allow students to engage in more profound types of learning. In the forthcoming evaluation study students will have the opportunity to inspect their student model, and we hope to show that the visualization of the student model will have a positive impact on student's learning.

ACKNOWLEDGEMENTS

We are grateful to the anonymous reviewers whose recommendations enabled us to significantly improve this paper. We thank Stellan Ohlsson for all the support and ideas, Kurt Hausler for programming support, and Brent Martin for helping with data analysis. We also thank our students, for putting their time and effort into trying out SQLT-Web and commenting on it. This research could not have been done without the support of other past and present members of ICTG. The work presented here was supported by the University of Canterbury research grants U6242 and U6430.

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. *International Journal of Artificial Intelligence in Education*, 10, 183-197.

Alpert, S., Singley, M., & Fairweather, P. (2000). Porting a Standalone Intelligent Tutoring System to the Web. In C. Peylo (Ed.) *Proceedings of ITS'2000 Workshop on Adaptive and Intelligent Web-based Education Systems, ITS'2000* (pp. 4-11).

Brusilovsky, P., Schwarz, E., & Weber, G. (1996). ELM-ART: an Intelligent Tutoring System on World Wide Web. In C. Frasson, G. Gauthier, A. Lesgold (eds.) *Proceedings of 3rd International Conference on Intelligent Tutoring Systems, ITS'96* (pp. 261-269). LCNS 1086. Berlin: Springer.

Brusilovsky, P. (1999). Adaptive and Intelligent Technologies for Web-based Education. *Kunstliche Intelligenz*, 4, 19-25.

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

Elmasri, R., & Navathe, S.B. (1994). *Fundamentals of database systems* (2nd edition). Redwood: Benjamin/Cummings.

Forgy, C.L. (1982) Rete: a Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence,* 19, 17-37.

Heift, T., & Nicholson, D. (2001). Web Delivery of Adaptive and Interactive Language Tutoring. *International Journal of Artificial Intelligence in Education*, 12(4), 310-324.

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

Mayo, M., & Mitrovic, A. (2000). Using a probabilistic student model to control problem difficulty. In G. Gauthier, C. Frasson and K. VanLehn (Eds.) In *Proceedings of 5th International Conference on Intelligent Tutoring Systems, ITS'2000* (pp. 524-533). LNCS 1839. Berlin: Springer,

Mayo, M., & Mitrovic, A. (2001). Optimising ITS Behavior with Bayesian Networks and Decision Theory. *International Journal of Artificial Intelligence in Education,* 12, 124-153.

Mallery, J.C. (1994). A Common LISP Hypermedia Server. In *Proceedings of 1st International Conference On the World Wide Web.*

Mitrovic, A. (1998a). A Knowledge-Based Teaching System for SQL. In T. Ottmann, I. Tomek (Eds.) *Proceedings of ED-MEDIA'98* (pp. 1027-1032). VA: AACE.

Mitrovic, A. (1998b). Experiences in Implementing Constraint-Based Modeling in SQL-Tutor. In *Proceedings of 4th International Conference on Intelligent Tutoring Systems, ITS'98* (pp. 414-423). LNCS 1452. Berlin: Springer.

Mitrovic, A. (2001) Investigating students' self-assessment skills. In M. Bauer, P.J.Gmytrasiewicz & J. Vassileva (Eds.) *Proceedings of 8th Int. Conference on User Modeling, UM-2001* (pp. 247-250). LNAI 2109. Sonthofen, July 2001. Berlin: Springer-Verlag.

Mitrovic, A. (2002). NORMIT, a Web-enabled Tutor for Database Normalization. In Kinshuk, R. Lewis, K. Akahori, R. Kemp, T. Okamoto, L. Henderson, & C-H Lee (Eds.) *Proceedings of ICCE 2002* (pp. 1276-1280). Los Alamitos, CA: IEEE Computer Society.

Mitrovic, A., Koedinger, K., & Martin, B. (2003) A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modelling. In P. Brusilovsky, A. Corbett, & F. de Rosis (Eds.) *Proceedings of the 9th International Conference on User Modeling, UM-2003* (pp. 313-322). LNAI 2702. Berlin: Springer-Verlag.

Mitrovic, A., & Martin, B. (2000). Evaluating the effectiveness of feedback in SQL-Tutor. In Kinshuk, C. Jesshope, T. Okamoto (Eds). *Proceedings of International Workshop on Advanced Learning Technologies, IWALT2000* (pp. 143-144). Palmerston North, New Zealand.

Mitrovic, A., Martin, B., & Mayo, M. (2001) Using Evaluation to Shape ITS Design: Results and Experiences with SQL-Tutor. *User Modeling and User-Adapted Interaction*, 12(2-3), 243-279.

Mitrovic, A., & Ohlsson, S. (1999). Evaluation of a constraint-based tutor for a database language. *International Journal of Artificial Intelligence in Education*, 10(3-4), 238-256.

Mitrovic, A., & Suraweera, P. (2000). Evaluating an Animated Pedagogical Agent. In G. Gauthier, C. Frasson, & K. VanLehn (Eds.) *Proc*eedings of *5th International Conference on Intelligent Tutoring Systems, ITS'2000* (pp. 73-82). LNCS 1839. Berlin: Springer-Verlag.

Ohlsson, S. (1994). Constraint-based Student Modeling. In *Student Modeling: the Key to Individualized Knowledge--based Instruction*, (pp167-189). Berlin: Springer-Verlag,

Ohlsson, S. (1996). Learning from Performance Errors. *Psychological Review,* 103(2) 241-262.

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.

Peylo, C., Thelen, T., Rollingen, C., & Gust, H. (2000). A Web-based Intelligent Educational System for PROLOG. In C. Peylo (Ed.), *Proceedings of ITS'2000 Workshop on Adaptive and Intelligent Web-based Education Systems* (pp. 62-68).

Ritter, S. (1997). PAT-Online: a Model-Tracing Tutor on the World-Wide Web. In P. Brusilovsky, K. Nakabayashi, S. Ritter (Eds.) AI-ED'97 Workshop on Intelligent Educational Systems on the World Wide Web (pp. 11-17).

Siekmann, J., Benzmuller, C., Fiedler, A., Franke, A., Goguadze, G., Horacek, H., Kohlhase, M., Libbrecht, P., Meier, A., Melis, E., Pollet, M., Sorge, V., Ullrich, C., & Zimmer, J. (2000). Adaptive Course Generation and Presentation. In C. Peylo (Ed.) *Proceedings of ITS'2000 Workshop on Adaptive and Intelligent Web-based Education Systems*, *ITS'2000* (pp. 54-61).

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

Stern, M., Woolf, B. P., & Kurose, J. F. (1997). Intelligence on the Web? In B. de Boulay, & R. Mizoguchi (Eds.) *Proceedings of 8th World Conference on Artificial Intelligence in Education, AI-ED'97* (pp. 490-497). Amsterdam: IOS.

Suraweera, P., & Mitrovic, A. (2002). KERMIT: a Constraint-based Tutor for Database Modeling. In S. Cerri, G. Gouarderes, F. Paraguacu (Eds.) *Proceedings of ITS 2002* (pp. 377-387). LNCS Vol. 2363. Berlin: Springer-Verlag.

Suthers, D., & Jones, D. (1997). An Architecture for Intelligent Collaborative Educational Systems. B.de Boulay, & R. Mizoguchi (Eds.) *Artificial Intelligence in Education: Knowledge and Media in Learning Systems* (pp. 55-62). Amsterdam: IOS.

Vassileva, J. (1997). Dynamic Course Generation on the WWW. In B. de Boulay, & R. Mizoguchi (Eds.) *Artificial Intelligence in Education: Knowledge and Media in Learning Systems* (pp. 498-505). Amsterdam: IOS.

Warendorf, K., & Tan, C. (1997) ADIS – An animated data structure intelligent tutoring systems or Putting an interactive tutor on the WWW. In P. Brusilovsky, K. Nakabayashi and S. Ritter (Eds.) *Proceedings of the Workshop on Intelligent Educational Systems on the World Wide Web at AIED'97* (pp 54-60). ISIR.

Weber, G., & Brusilovsky, P. (2001). ELM-ART: An Adaptive Versatile System for Web-based Instruction. *International Journal of Artificial Intelligence in Education*, 12(4), 351-384.

APPENDIX A: USER QUESTIONNAIRE (STUDY 2)

1. What is your previous experience with SQL?
   a) only lectures b) lectures plus some work c) extensive use
2. How much time did you need to learn about the system itself and its functions?
   a) most of the session
   b) 30 minutes
   c) 10 minutes
   d) less than 5 minutes
3. How much did you learn about SQL from using the system?
   Nothing Very much
   1 2 3 4 5
4. Did you enjoy learning with SQL-Tutor?
   Not at all Very much
   1 2 3 4 5
5. Would you recommend SQL-Tutor to other students?
   a) Yes b) Do not know c) No
6. Do you find the interface easy to use?
   Not at all Very much
   1 2 3 4 5
7. Do you find the display of the schema understandable?
   a) Yes b) Do not know c) No
8. Do you find feedback useful?
   Not at all Very much
   1 2 3 4 5
9. Would you prefer more details in feedback?
   a) Yes b) Do not know c) No
10. How often did you use ``System's Choice'' to have the system select a problem for you to solve?
    Never Always
    1 2 3 4 5
11. If you have used ``System's Choice'', how do you rate the difficulty of the problems SQL-Tutor selected for you?
    Always too easy Always too hard
    1 2 3 4 5
12. Did the problems selected by ``System's Choice'' target SQL concepts that you feel were appropriate at the time? Please comment.
    Never appropriate Always appropriate
    1 2 3 4 5

13. Did you feel that the problems selected by ``System's Choice'' were better or worse than those that would have been selected by a human tutor?
    Always worse Always better
    1 2 3 4 5
14. Did you encounter any software problems or crashes?
    a)Yes b) No
15. What do you like in particular about SQL-Tutor?
16. Is there anything you found frustrating about the system?