ERM-Tutor an ER-to-Relational Mapping Tutor

BSc. Honours Research Report

Melinda Marshall, Dr. Antonija Mitrović (supervisor)

Department of Computer Science & Software Engineering
University of Canterbury
December 11, 2004

Abstract

Relational Databases are extremely commonplace in today's society. Because of this the skill of designing such databases is often taught in university courses, where part of this skill involves being able to map from a conceptual schema to relational database tables. This is a mapping algorithm that maps from the Entity Relationship conceptual schema to relational database tables, and applying this algorithm is an area in which student errors often occur. One-to-one human tutoring is the most effective mode of teaching, however this is often not feasible. Intelligent Tutoring Systems aim to simulate a human tutor by providing individualised tuition, and can be effective in place of a human tutor.

ERM-Tutor is an *E*ntity *R*elationship *M*apping Constraint-Based Intelligent Tutoring System developed to provide a practice environment to map entity relationship diagrams using the mapping algorithm. ERM-Tutor has a problem set of Entity Relationship diagrams, which students map to relational database tables by being led sequentially through the seven steps of the algorithm. The students are given feedback on their solutions after each table they map, and at the end of each step of the algorithm. A preliminary evaluation carried out on ERM-Tutor provided favourable results that suggested that the tutor was effective in helping students learn. However this will need to be followed by a more formal evaluation.

Contents

1 Introduction

This paper presents ERM-Tutor, a constraint-based ITS for mapping Entity Relationship (ER) diagrams to Relational Database Schema. It is a problem solving environment in which a student has to follow a series of steps to systematically map different elements of the ER diagram into tables using the algorithm described in The Fundamentals of Database Systems [8].

The introductory chapter of this report has a high level look at ITSs in Section 1.1. ERM-Tutor was developed to address the problem that students have with learning to apply the mapping algorithm, and whether an ITS could be a practical solution. This problem and our solution is described in detail in Section 1.2. The remaining Section 1.3 outlines the structure of the remainder of the paper.

1.1 Intelligent Tutoring Systems

Intelligent Tutoring Systems are developed with the goal being to emulate a human teacher using artificial intelligence techniques. The distinction between an intelligent tutor and just a computer based tutor comes from the fact that ITSs attempt to individualise tuition, as a human tutor would, by using techniques such as customising feedback on students' solutions, selecting problems based on a student's current aptitude, or ensuring the student always remains on the correct path to the solution.

In order to individualise tuition, the ITS must dynamically reason about the domain, the student and specifically their knowledge, the teaching strategy, and the communication strategy. Two very important aspects to an ITS therefore become the student model and the domain knowledge. Because tuition in an ITS is individualised, it becomes immediately obvious that the tutor therefore needs to distinguish between different students, and knows some information about each student, in other words maintains a student model. The student model is ideally a current and accurate picture of the student's knowledge which includes correct and incorrect knowledge, and allows the tutor to reason about each student. The domain knowledge provides a standard for evaluating a student's solution, and also is the source of knowledge that the tutor is teaching.

In most cases ITSs provide students with a problem solving environment so the student is learning cognitive skills by doing. They are therefore allowing students to practise a task, and practice is always necessary when acquiring instrumental knowledge, that is knowledge about how to perform tasks [24]. It is therefore in domains where a student must gain knowledge about either attaining goals, achieving effects or performing tasks that ITSs are ideal. An ITS will characteristically provide a set of problems, an interface tailored to solving problems of that nature, and give the student a critque on their solution.

ITSs have been successfully developed for a variety of domains. A tutor developed for teaching students the LISP programming language called Lisp-Tutor [1] had a positive result. Using the Lisp-Tutor students gained the same increase in knowledge in 30% less time and learning was improved by 1 standard deviation. Sherlock is a tutor for Electronics Troubleshooting that had the result of 20-25 hours of using the system had the same effect of four years on-the-job experience [14]. There was an increase of 0.9 of a standard deviation found by students using Atlas, a Mixed-Initiative, Multimodal Tutoring System [11].

1.2 Motivation for an Entity Relationship Mapping Tutor

Databases are extremely commonplace in today's society, consequently there is a demand for well designed databases. Techniques and models have been developed to aid the creation of elegant databases, including the Entity Relational Model. The entity relationship model is a popular high-level conceptual data model thich was proposed by Chen [7], and is used as the first step in designing databases. This first step involves creating a diagram where the basic element is an entity which represents an independent object that exists in the real world. These entities have attributes which are properties that belong to it, for example a student entity may have a student-number attribute. Entities in this conceptual model are linked to each other with relationships [8].

The Entity-Relational model provides a high-level design of a database. The relational database schema is a lower-level representation of a database as a collection of relations, where a relation can be thought of as a file of related records. The relational model is very popular with major commerical success, and the SQL language for relational databases has a standard formalising the data model [8]. It is therefore important that Entity-Relationship Diagrams can be mapped to relational database tables. This process is called logical database design, or data model mapping, and there is an algorithm to carry this task.

The algorithm to map from Entity-Relationship diagrams to relational tables is often taught in university database courses. Despite the fact that it is a procedural algorithm, mapping is still a skill in which students make mistakes. Also the lecture environment is not the ideal situation for learning a problem solving skill as it does not allow the student guided practice. One-to-one human tutoring has been empirically proven to be more effective than traditional classroom teaching, with learning performing improvements of two standard deviations [6]. Although some university courses offer tutorials to supplement lectures, these are limited by a high student:tutor ratio and still cannot provide full individualised tuition.

Intelligent Tutoring Systems (ITS) have the goal of simulating a human tutor, so providing one-to-one tuition can be feasible in domains where it has not been previously. ITSs have been shown to be effective in increasing students' learning performance in domains where students require practice to become proficient [21]. ERM-Tutor is an *E*ntity *R*elationship *M*apping ITS developed to provide a practice environment for the skill of mapping Entity Relationship diagrams to relational database tables. The tutor teaches the skill adhering precisely to the mapping algorithm described in [8] supplying the student with mapping problems to solve. The main aim of the tutor is to individualise tuition to each student, with the supplementary goal of investigating whether mapping is a domain appropriate to Constraint Based Modelling [23].

A preliminary evaluation of ERM-Tutor evaluated its effectiveness and contribution to learning. This study involved voluntary participants using the tutor for a variable amount of time. The students carried out a pre-test upon logging onto the tutor, and could use ERM-Tutor for as much or as little time as they desired. The students were enrolled in an introductory database course, and the examination for this course was used a post-test. The examination results for the students who used the tutor were significantly higher than for the students who did not use ERM-Tutor. We can not conclude that the tutor alone contributed to these results, but it is still a positive result. The learning curves generated from the student logs of tutor users also supported the proposition that ERM-Tutor helps students learn.

1.3 Report Structure

The architecture and description of ITS components is described in Chapter 2. The history of student modelling techniques is explained, and in particular Constraint Based Modelling is discussed, as this is the modelling technique used in ERM-Tutor. Chapter 2 also describes some other ITSs that have been developed in the database domain, and Entity Relationship mapping algorithm is described in depth. ERM-Tutor is introduced in Chapter 3, including a detailed description of its design and implementation. The preliminary evaluation of ERM-Tutor that was carried out at the University of Canterbury is detailed in Chapter 4. Chapter 4 also describes the results of this evaluation. Finally, the conclusions are presented in Chapter 5, which also discusses the future work for ERM-Tutor.

2 Background

2.1 Intelligent Tutoring Systems

Computers have had an increasing role in the field of education since computer-based training and computer aided instruction were introduced over thirty years ago. The first generation of computer tutors were focussed on the presentation of domain knowledge, similar to an electronic book. Since then the systems became more adaptive with more flexible navigation and hypermedia included. These computer tutors have now evolved to Intelligent Tutoring Systems. An ITS has the goal of providing one-to-one teaching without having to have a human tutor. Intelligent tutoring systems have been shown to be effective and improve students' performances by 0.3-1 standard deviation, and increase students' motivation.

ITSs can be implemented in a variety of ways, but most can be decomposed into several common components. These are the student model, the pedagogical module, the domain knowledge, the communication module and the expert module. In addition to these five components there is also often a student modeller that creates the student models, an interface using the communication module, domain knowledge used by the domain knowledge module. See Figure 2.1 for the interactions of the five main ITS components.

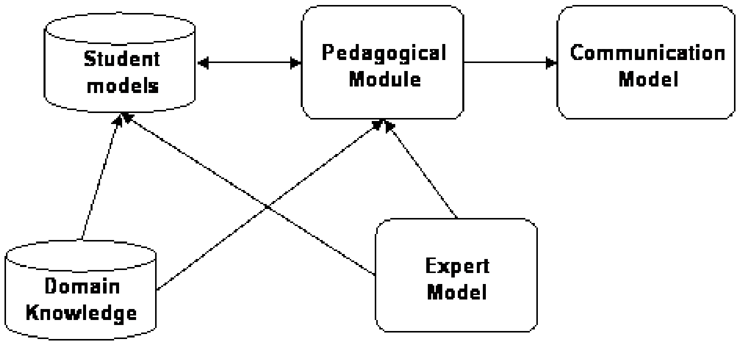


Figure 2.1: Architecture of a Typical Intelligent Tutoring System.

2.1.1 Student Model

The student model is a vital part of allowing ITSs to tailor tuition to the individual student. It is a representation maintained by the tutoring system of what it knows about the learner. One way of recording information about the learner would be to record the student's history and then draw inferences. However, it is difficult to do this retrospectively so a student model would normally draw the inferences online. [13] A comprehensive student model would contain the following aspects: [13]

- All prior relevant learning.
- The learner's progress within the curriculum.
- The learner's preferred learning style.

- Other learner related information.

However, such a model is impossible to create and maintain, and it is not essential that a student model be that detailed to effectively tutor the student [25].

Because it is such a vital part of ITS student modeling is an area of ITS in which a variety of solutions have been researched. In this section we will briefly describe overlay models, perturbation and bug models, and model tracing, and constraint-based modelling will be described in more detail in Section 2.1.1.

Overlay models treat the student's model as a subset of an expert's knowledge. Overlay models assume that all differences between a student's knowledge and experts' knowledge can be explained by a student's lack of skill. There have been some successes with Overlay models: SimTutor is a tutor for simulation modelling that found students' performances increased after using the tutor [4]. Overlay models do work well where the goal is to transfer knowledge to the student. However, an overlay student model is not a complete picture of the student's knowledge. This type of model does not account for any knowledge the student might have that is different to the expert knowledge. In other words, an overlay model does not account for misconceptions the student might have.

The lack of representing misconceptions in overlay models has been addressed by perturbation models and bug models. A perturbation model is an enhanced overlay model that adds a representation of incorrect knowledge. The incorrect knowledge is stored in a similar way to the correct knowledge in the knowledge base. Similar to misconceptions being stored in a knowledge base, bugs can also be stored in a bug library. Bugs are distinct from misconceptions as a bug is a flaw in a procedure that manifests itself in faulty behaviour. A bug is an explanation as to why a mistake has occured, whereas a perturbation model storing misconceptions does not give this information. Bug libraries that contain an accurate model of bugs and maintain a good domain coverage are extremely difficult to create [13].

Model tracing is a second generation modelling technique. Model tracing is based on the ACT-R [3] (Adaptive Cognitive Theory, rational analysis extension) theory of human cognition which assumes a rule-based cognitive architecture. In a production system there are two types of knowledge in the long term memory: declarative knowledge and procedural knowledge. Declarative knowledge is factual knowledge acquired from reading or instruction, and procedural knowledge is made up of production rules. A production rule consists of three parts: a goal, a situation and an action. For example, a very simple production rule for lighting a room can be seen in Listing 2.1.

```
Goal = Want more light
Situation = Light off
Action = Turn on light
```

Listing 2.1: Example Production Rule

The main idea behind model tracing is to follow the student's solution path and check their solution at each step. To analyse the student's solution requires a set of production rules, both correct rules gained from analysing experts, and incorrect rules that can be generated from analysing the student's solutions. As the student makes an action, a model tracing system will match the student's behaviour against the set of incorrect and correct rules. If the student's action matches an incorrect rule, the tutor will print out the feedback message associated with that rule, otherwise it will remain silent [3]. This approach is limited by the completeness of the bug library, and the fact that there is only a limited number of solutions or actions at each step considered correct and or while creative solutions are not accounted for.

Constraint-Based Student Modelling

Constraint-Based Student Modelling is a student modelling approach that does allow creative student solutions. Constraint-Based modelling was developed by Stellan Ohlsoon [23] and is based on the theory of learning by performance errors [24]. The theory states that skills are learnt by practising, and the learner receiving feedback on the performance. There are two types of feedback the learner can be given: postive feedback and negative feedback. Postive feedback is information about successful actions, and negative feedback is information about an incorrect action the learner performed. It is this negative feedback that a learner will use to restrict a faulty ruleset to applications in which it does not cause errors. As well as this error correction mechanism, a learner also has an error detection mechanism where they compare their solutions with correct ones and together these form the basic learning process. Both error detection and error correction can be represented as declarative knowledge in the form of constraints on correct solutions.

Constraints have the property of dividing the solution set into two equivalence classes: solutions that violate the constraint and solutions that do not. Constraint-based modelling follows the theory that it is the problem state that is where the important diagnostic information is, unlike model-tracing, which believes it is in the sequence of student actions. Constraints can be abstracted beyond individual problem solving steps, and because the set of incorrect knowledge is so vast it is necessary to abstract constraints to a level where the basic principles are covered. In this way the domain knowledge for the ITS can be built from constraints that can be grouped by their pedagogical importance to the tutor. Constraints are not bug libraries but they do model errors indirectly.

A constraint is composed of two main parts: the relevance condition which is the class of solutions for which this constraint is relevant, and the satisfaction condition. They can be written in English as follows: *If <relevance condition> is true, then <satisfaction> had been also be true, otherwise something has gone wrong*. An example from the mathematics domain of fractions is as follows: If the current problem is $n_1/d_1 + n_2/d_2$ and if $n = n_1 + n_2$ then it had been be the case that $d_1 = d_2$ (or else something has gone wrong [23]. This is a constraint with two parts to the relevance conditions stating that the student is adding fractions *and* the solution for the numerator is equal to the sum of the two numerators.

Constraints can be implemented in a variety of ways and the most common way is to represent constraints as pairs of patterns. When implemented in this way, evaluation of constraints against a problem state becomes a very efficient task computationally as any standard pattern matching algorithm can be applied. The RETE [9] algorithm is a pattern matching algorithm that has successfully been applied to constraint evaluation in several ITSs, for example SQL-Tutor [17] and NORMIT [18].

Constraint-Based ITSs also have the advantage that a knowledge base written in constraints takes significantly less time than writing production rules. In fact an English language skills tutor, LBITS, had a constraint set of 315 constraints and was written in just five hours [16]. Building a domain model using constraints has also been shown to be a surmountable task for novice intelligent tutoring system authors [15]. An experiment conducted at the University of Canterbury found that graduate students taking part in an Intelligent Tutoring Systems Paper were able to create domain models for teaching students how to write the plural form of English words [15]. The domain models were similar to ones in LBITS that the students had access to. However, it was still a good result for novices.

2.1.2 Pedagogical Module

The pedagogical module is the component of the tutoring system that performs the teaching functions. It is very dependent on the student model, as this often dictates the pedagogy of the system. The pedagogical actions of the system can be diagnostic which includes updating or forming the student model, or didactic where they are conveying instruction and information. Examples of the pedagogical role include, deciding when to review information, when to present new information and which topic to present to the student next. [5] Another important role of the pedagogical module in many ITSs is the presentation of feedback on a student's solution, including when to give feedback and at which level. Some pedagogical strategies include enforcing correctness, computer coach, socratic teaching, and collaborative learning.

The pedagogical strategy of enforcing correctness can be seen in tutoring systems that use model tracing. As well as being a student modelling technique (see Section 2.1.1), model-tracing is a pedagogical strategy in which the computer system has full control over the teaching. The student is not allowed to take more than one step wrong in solving a problem; as soon as they make an error, they get immediate feedback on their action. This style of teaching based on the ACT theory is efficient for novices because they do not spend time floundering and being off-track [3]. However, this style is also very restrictive, not allowing alternative paths to a correct solution and may therefore be frustrating for experienced students.

The computer coach pedagogical strategy is more likely to appeal to experienced students. A computer coach is a strategy that lets the student be in control of their learning. The system is normally quiet unless the student requests feedback or needs help from the tutoring system. SQL-Tutor [17], ER-Tutor [27] and Normit [18] are all examples of computer coaches. In all of these tutors there is a button a student can click on to submit their solution. It is only after clicking this button that the tutor will evaluate the student's solution in any way and give them feedback. A variation of SQL-Tutor included an animated pedagogical agent that enhanced the computer coaches role to give the student help or encouragement if they appeared to be floundering, or idle for an extended period [26]. Without the ability to offer help computer coaches can be frustrating for novice users, and conversely, with that ability they can be frustrating for experienced users.

The last two pedagogical strategies are socratic teaching and collaborative learning. Socratic teaching is an approach where the tutor poses questions and counter examples to the student so that they can form general principles. The advantage of this pedagogical strategy is that it allows the student a free exploration of the problem space, but like computer coaches it can be frustrating for novice users. Collaborative learning is where students work together on solutions, or have access to each others' solutions to help them form their own.

2.1.3 Interface

The interface of an ITS is an extremely important component as it will dictate the subjective satisfaction a student will feel when using the tutor. If the interface is difficult to use, flawed with bugs, or just unpleasant to look at this may discourage the student from using the tutor again. The interface can be relied upon to uphold some important principles of ITS design.

Minimizing the student's working memory load is one such design principle for ITSs that is the responsibility of the user interface. As a student is trying to learn a piece of domain knowledge, all the relevant information must be kept simultaneously in working memory. The user interface can help with this by minimising the presentation of information while the student is problem solving [3]. Another way in which the interface can reduce the user's working memory load is to make sure all the information they need in a problem solving environment is easily visible, or available to them, to prevent them having to keep information in their memory.

The second design principle that is strongly related to the user interface is that a tutoring system should communicate the goal structure underlying the problem solving [3]. This can be done by designing the interface so that the immediate goals the student needs to work on are made clear. For example in SQL-Tutor the student is learning to write SQL select statements and when writing such statements there are a number of different clauses that may need to be included [17] . So some goals the student should concentrate on are "select the correct attributes from the correct tables", and "specify the correct tables we use for this query". SQL is normally written in one piece of text. However to help the student visualise the goals they need to complete, the interface divides the task into text input boxes for each clause, or goal, in the problem. This has the effect of structuring the student's thinking.

Another vital goal an ITS interface should have is that it should facilitate the conceptualisation of the domain knowledge. The way this is done is very much domain dependent; for example in a geometry domain the interface should provide the student with appropriate graphical diagrams. In the domain of creating Entity Relationship Schemas the domain is very graphical so it would be unnatural to make students solve problems in a textual manner. Instead ER-Tutor allows the student to construct answers using graphical components [27]. To be consistent, the solutions for problems are also presented graphically.

2.1.4 Domain Knowledge and Runnable Expert Module

The domain knowledge is the knowledge or skills that the tutor is aiming to impart to the students. The domain performs two functions in an ITS: providing the source of knowledge to teach; and providing the standard for evaluating a student's solution. One difficulty with building a knowledge base is designing a representation of knowledge so that it can be accessed by the various components of the tutor that use it [5]. It is because of this that a lot of research in this domain knowledge is focussed on representing it, and whether this can be done in such a way as to represent concepts and mental models.

An Expert Module contains the same knowledge as the domain knowledge, but with the difference that the expert module is a model of how an expert would represent knowledge in the domain. Often the expert module is a runnable model of the domain, in other words a problem solver. The information from the expert module can be used in the same way as domain knowledge [5].

The way the domain knowledge is represented can be influenced very strongly by the student modelling, or pedagogical strategy employed by the tutoring system. For example a model-tracing tutor will store the domain knowledge in a knowledge base of production rules, including buggy rules. A constraint based tutor can store all its domain knowledge in a constraint base, where the constraints also form the student model.

In some cases there can be a mixture of both a knowledge base and an expert module. An example is Normit, a constraint-based tutor for teaching database normalisation. The knowledge base is in the form of a constraint base containing 54 constraints. However, Normit also has a problem solver. The problem solver gives the tutor the ability to generate a solution for any normalisation problem, and so the semantic constraints in the system compare the student's solution to the ideal solutions generated by the problem solver [18].

2.2 Entity-Relationship to Relational Database Mapping

Designing and creating a database from user requirements is a complex task consisting of many steps. When creating a relational database, the first step in the process is to create a conceptual schema design. This can be in the form of an Entity Relationship, or Enhanced Entity Relationship diagram. This conceptual schema describes the structure of the database, describing entities and relationships and hiding the physical structure of the database.

The conceptual schema is a very good abstraction for visualising the database, but to go a step further to implementation, a relational database schema must be defined. The relational database schema defines the relations, their attributes and keys. To create a relational schema, you must map concepts from the Entity Relationship or Enhanced Entity Relationship schema to concepts of the relational model. This mapping process is called data model mapping, or logical database design.

There exist mapping algorithms to carry out the logical database design task. The ERM-Tutor was developed to teach students the ER-to-Relational Mapping Algorithm. This is a seven step, procedural algorithm. Each step in the algorithm maps one concept from the Entity Relationship diagram by either creating a new relational database table, or altering previously created relational database tables by adding foreign keys and attributes [8].

Step 1 Regular Entities

The first step in the algorithm involves creating a new relational database table for each regular entity in the schema. Each simple attribute of the entity is added to the relation, and one of the candidate keys from the entity is added as the primary key of the new relation. If the attribute is a composite attribute then you just add the component attributes, and if it is a multivalued attribute it is not included. In our example Figure 2.2 "PUBLICATION" is an example of an entity that would be mapped at this step.

Step 2 Weak Entities

A new relation is also created for each weak entity. The attributes of the weak entity are included in a similar way to the regular entities. The primary key of this relation consists of the partial key, and a foreign key from the owner entity. Any attributes of the identifying relationship should also be included in the new table. For example the "SUBPROJECT" weak entity from the example diagram would be mapped at this step.

Step 3 1:1 Binary Relationships

For step 3 no new table is created, instead choose to alter one of the tables that corresponds to the entities participating in the relationship. If one of the entities participates in the relationship totally, the table corresponding to this entity should be the one you alter. The primary key of the table you are not altering needs to be added as a foreign key, and any attributes of the relationships should also be added. The "SUPERVISED_BY" relationship in Figure 2.2 is a relationship that step 3 maps.

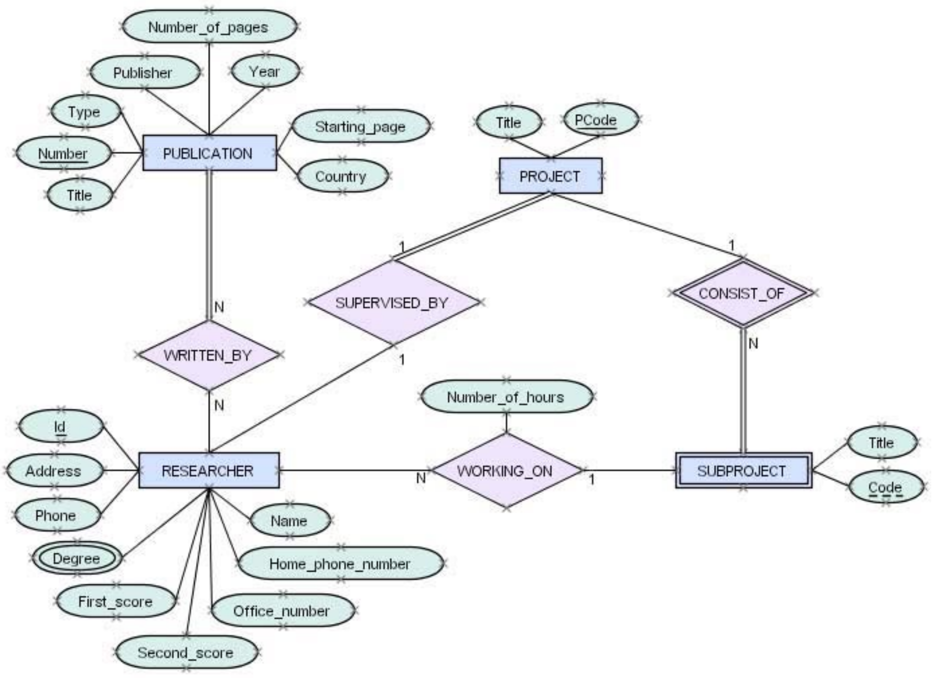


Figure 2.2: An Entity Relationship Diagram for a Projects Database.

Step 4 1:N Binary Relationships

When mapping binary 1:N relationships we again alter one of the tables corresponding to an entity participating in the relationship. This time we chose the table that participates with a cardinality of N to alter. The key of the table that was mapped from the other entity should be included as a foreign key in the table you are altering. Again the attributes of the relationship are added to the table you altering. From the example diagram at this step we would map the “WORKING_ON” relationship. The “CONSISTS_OF” relationship would not be mapped as this is an identifying relationship, and covered by step 2.

Step 5 M:N Binary Relationships

A new relational database table is created for each M:N relationship. The primary key of this new table is made up of two foreign keys, these are the primary keys of tables that participate in the relationship. The attributes of the relationship are added to the new table. In the example diagram there is only relationship that needs to be mapped at this stage, and this is the “WRITTEN_BY” relationship.

Step 6 Multivalued Attributes

For each multivalued attribute a new relation created. The attribute itself is added to this table as a primary key, also added as a primary key and foreign key is the primary key from the table relating to the entity that the multivalued attribute belongs to. If the multivalued attribute is composite, only the component attributes are added. The “Degree” attribute would be mapped from our example at this step in the algorithm.

Step 7 N-ary Relationships

An N-ary relationship is one that has more than two entities participating in it and in this situation, a new table is always created. The primary key is made up of the primary keys from the tables corresponding to each entity participating in the relationship, and they are added as foreign keys. There can be exceptions to this however: if an entity participates with a cardinality of 1, then a foreign key should not be added from this table. There are no N-ary relationships in our example diagram (Figure 2.2).

Alternative Mappings Techniques

The decribed algorithm is not the only way in which Entity Relationship diagrams can be mapped. There are alternate strategies for mapping the binary 1:1 and 1:N relationships. The foreign key approach for mapping those relationships can be replaced by creating a new relational table as is done in step five of the algorithm. This approach is appropriate when there are are few instances of the relationship, in this situation using the foreign key approach would result in a lot of null values in the foreign attribute of the table [28].

3 Design and Implementation

ITSs have proven to be effective in many domains and constraint-based tutors in particular have been shown to enhance learning in a variety of domains [21]. ER-Tutor [27] and SQL-Tutor [17] are two constraint-based tutors in the database domain, and Normit [18] in particular is a tutor for a procedural database domain, normalisation. There have been no other successful tutoring systems for entity relationship to relational database mapping developed, and so the ERM-Tutor system was developed. The goal was to see if constraint-based tutoring could be applied to Entity Relationship mapping to produce an effective tutor.

This chapter discusses the design and architecture of ERM-Tutor. Section 3.1 gives an *overview* of the teaching system describing how the mapping algorithm is broken into small tasks for the student to solve. The *architecture* of the system and how the different components relate is described in Section 3.2. Section 3.3 describes the components of the *user interface* of ERM-Tutor. A description and example of the *problems* and their different *representations* is given in Section 3.4. Because the mapping task has a procedural alogrithm that dictates how things should be mapped, it was possible to create a Problem Solver for the tutor, eliminating the need for manually created ideal solution, as described in Section 3.5. The *Knowledge Base* for ERM-Tutor is discussed in Section 3.6. The *student modeller* and *pedagogical module* are described in Section 3.7 and Section 3.8.

3.1 Overview

ERM-Tutor is a constraint-based ITS designed to tutor students in the skill of mapping entity relationship diagrams to relational database tables. The mapping algorithm it strictly adheres to is the algorithm as detailed in Fundamentals of Database Systems [8]. The tutor is a problem solving environment for individual tuition. It is not a teaching system and so is designed to be a supplement to learning as would a human tutor. This means it does not give the student any domain knowledge other than through feedback messages. It is assumed the student will already be familar with entity relationship diagrams, relational database tables, and the mapping algorithm between the two.

ERM-Tutor is web-based and so the system is accessed through a web browser to view web pages. First the student is required to log into the system which will load their student model. The student can choose any problem to work on. The problems in the system are entity relationship diagrams which are to be mapped into relational database tables. The problem is broken into seven tasks, each corresponding to a step in the mapping algorithm, and presented to the student in the same order as the algorithm. An example of a task would be step one of the algorithm “Map all regular entities”. The interface provides the student with the working area to create or alter one relational database table at a time, therefore the task can be seen as being broken into subtasks. So this means for step one the student has the opportunity to map one regular entity, and the system provides the student with a means to check this single entity before moving on to map a second entity. The interface displays to the student a record of each correct table they have created or altered for that step, and when the student believes they have mapped all the necessary components for that step they can check all their tables. It is upon having a correct solution for the “check all tables” task that the student is moved on to the next task.

ERM-Tutor is implemented in Allegro Common Lisp and runs on the AllegroServe web server, an extensible web server provided with Allegro Common Lisp [10]. It is best viewed on Internet Explorer 5.0 or higher.

3.2 Architecture

There are seven main components to the architecture of ERM-Tutor and these can be classifed as either active components or system data. Included in the active components are the student modeller, the pedagogical module, the problem solver, and the interface. Supporting these active components are the knowledge base, the student models and the problem set. The way these components interact can be seen in Figure 3.1.

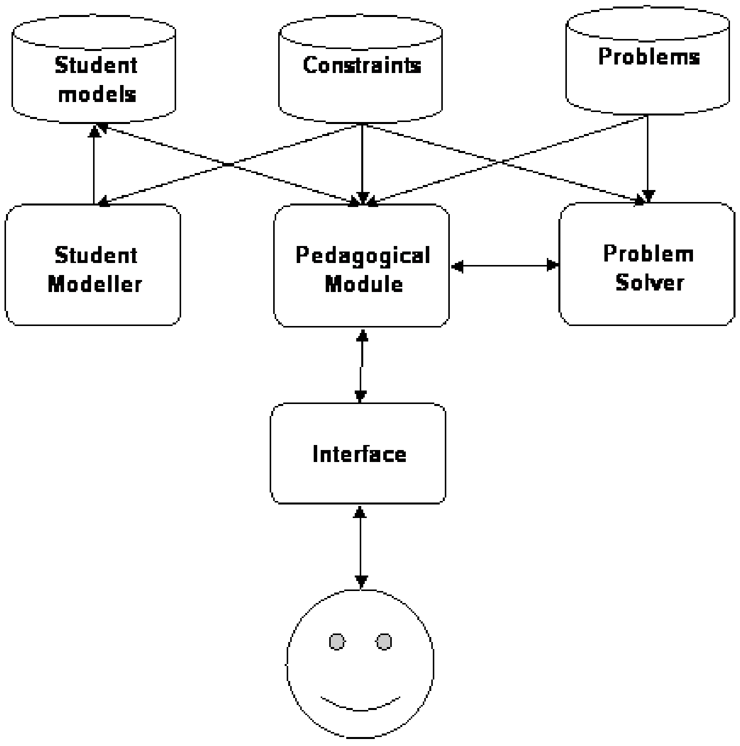


Figure 3.1: Architecture of ERM-Tutor.

3.3 User Interface

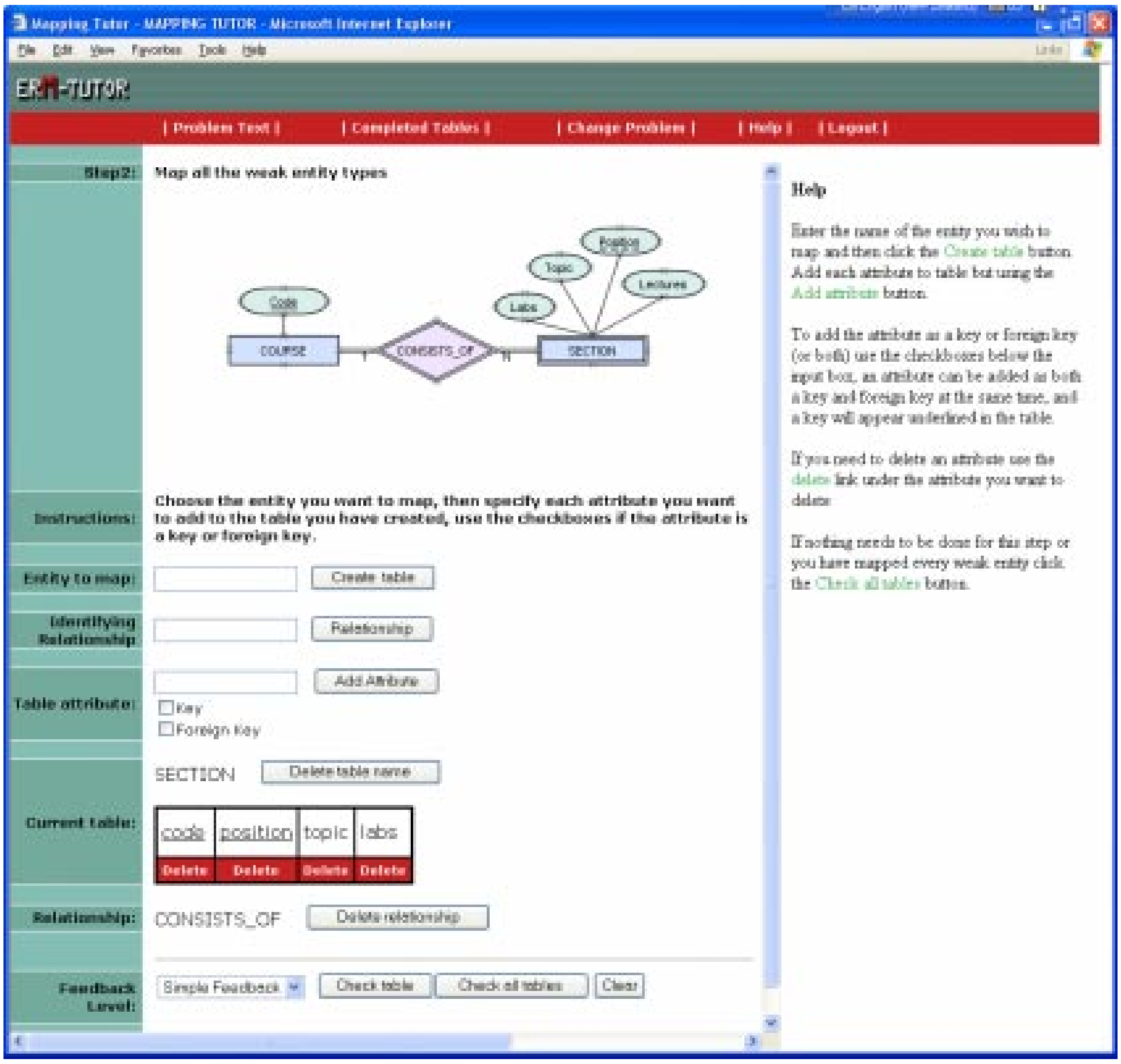
ERM-Tutor is a web-based system therefore the user interface takes the form of interactive web pages. There are five distinct types of web pages in the system:

1. The login page
2. The information page
3. The “choose problem” page
4. The task, problem solving page
5. The logout page

The login page simply provides a screen for the user to enter their name and password. They are then taken to the information page which gives the user instructions on how to use the system. They click on a “continue” button to navigate to the“ choose problem” page. Here the tutor presents a list of problems with their numbers and titles, either of which can be clicked on by the student to begin solving the problem. The problems titles are all initially in a black font. However, when the student completes a problem, its title will then be rendered in green to signify that it has already been correctly solved.

When a student selects a problem, they are taken to the page to solve the step one task. On successful completion of step one, they are taken to step two, and this continues until they complete step seven. When they complete step seven, the entire problem is taken to be correct and the student model is updated accordingly. For some problems, there is no action that needs to be taken for many of the steps. However, the student is still led through every step as a part of the skill of mapping is recognising when a step needs to be applied as well as when it is not relevant.

The main problem solving page can be seen in Figure 3.2 and is described in the following subsections.


Mapping Tutor - MAPPING TUTOR - Microsoft Internet Explorer
File Edit View Favorites Tools Help
ERM-TUTOR
| Problem Text | | Completed Tables | | Change Problem | | Help | | Logout |
Code
COURSE
CONSISTS_OF
SECTION
Labs
Topic
Position
Lectures
1
N

Step2: Map all the weak entity types

Instructions: Choose the entity you want to map, then specify each attribute you want to add to the table you have created, use the checkboxes if the attribute is a key or foreign key.

Entity to map: [Create table]

Identifying Relationship [Relationship]

Table attribute: [Add Attribute]
☐ Key
☐ Foreign Key

Current table: SECTION [Delete table name]

code	position	topic	labs
Delete	Delete	Delete	Delete

Relationship: CONSISTS_OF [Delete relationship]

Feedback Level: Simple Feedback [Check table] [Check all tables] [Clear]

Help

Enter the name of the entity you wish to map and then click the Create table button. Add each attribute to table but using the Add attribute button.

To add the attribute as a key or foreign key (or both) use the checkboxes below the input box, an attribute can be added as both a key and foreign key at the same time, and a key will appear underlined in the table.

If you need to delete an attribute use the delete link under the attribute you want to delete.

If nothing needs to be done for this step or you have mapped every weak entity click the Check all tables button.

Figure 3.2: User interface of ERM-Tutor.

3.3.1 Button Frame

The button frame is the same across each of the task pages. It presents the student with five buttons:

- *Problem Text* This button pops up a new window where the student can see the full textual description of the database requirements for the problem.

- *Completed Tables* This button pops up another window where the student can see all the tables they have completed for this problem over all the steps. This is necessary because in some steps you need to go back and alter a table, or add a foreign key so this allows students to view the design of the existing tables.
- *Change Problem* This button is a link to the change problem page. This is the means by which a student can abort a problem they are working on if they become frustrated or bored. The problem which they are currently working on will not be saved if they click this button.
- *Help* This button displays the help for the current page in the feedback/help area.
- *Logout* This button logs the student out. This involves saving the student model, and taking the student to the logout page.

Figure 3.3: The buttons available to the student while working on an ERM-Tutor task.

3.3.2 Problem Solving Area

The problem solving area is the main part of the page. It differs slightly for each step but the general layout is the same. First there is a short description of the student's task for that step. For example, for step two the task text reads "Map all weak entities". This is basically to remind the student what is required at this step, rather than be educational material in its own right. Next they are presented with the problem diagram, underneath which is short instructions on how to use the input boxes to create or alter a table.

The next aspect of the problem solving area is the input boxes. If the step requires that a table be created or altered, there is a box for the student to input the table name, and a corresponding button for the student to click when they want to submit the name. If the step requires that a relationship is mapped, there is an input box for the student to specify which relationship they are currently working on, and again a corresponding button. Lastly, there is always a box for the student to add attributes. They type the attribute name in the input area. Before they click a button to submit the attribute, they can choose if they want the attribute to be added as a key or foreign key (or both) by using provided checkboxes.

The next section of the problem solving area displays the solution the student is currently working on. It displays the name of the relationship they are mapping if there is one. The name of the relationship and the name of table both have corresponding "delete" buttons adjacent to them so the student can remove them if they want to alter their solution. It also displays the relational database table the student is creating or altering. This is presented to the student in the standard form with each attribute in adjoining rectangular boxes, and the keys underlined. However, the standard form does not have a facility for specifying which attributes are foreign keys, so we enhanced the display to show foreign keys in italics. To allow the student to individually select attributes to delete, there is a red box under each attribute with a "delete" button. This can be seen in Figure 3.4. The remaining part of their solution that is displayed is the list of elements that the student has already mapped for that step. For example, in step one the list of regular entities they have mapped is shown and for step three the list of relationships they have mapped is shown.

The last part of the main task page provides the student with the facility to get feedback on their solution. First is the feedback level selection combo box that has a dropdown menu for the different levels of feedback available to the student. Next to this is the "Check Table" button which the student can use to check the current relational database table they are working on. The "Check All Tables" button is next and the student will click on this to check that they have altered all created all the necessary tables for that step. The "Clear" button is the last one available to the student and this clears the current table the student is working on. When the student clicks the "Check All Tables" button and their solution is correct, the feedback box, and other buttons in this area are replaced with a "Done" button that the student can click on to move to the next step. In the case of step seven the button reads "Next Problem" and will take the student back to the "choose problem" page.

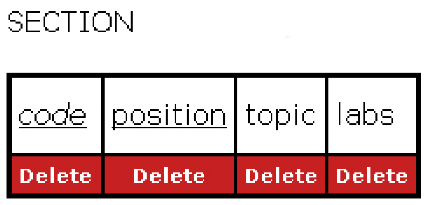
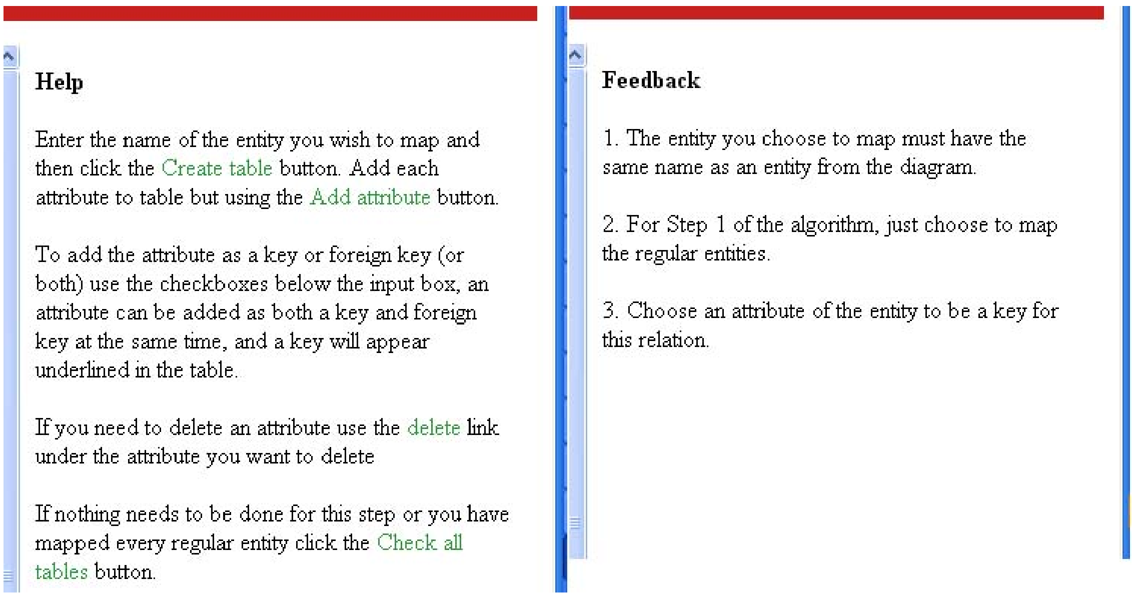


Figure 3.4: The display of a relational database table in ERM-Tutor's interface.

3.3.3 Feedback/Help Area

The feedback/help area occupies the right side of the screen. When a task page is first displayed to a student, the help page is displayed to the student by default. The help page provides a textual description of how to use the interface, each task has its own specific help page. When the student requests feedback by clicking on either the "Check Table" or "Check All Tables" buttons, the help page is replaced by the feedback. The help page can be redisplayed and replace the feedback page at any time by the student' clicking on the "Help". See Figure 3.5 for an example of the feedback and help areas.



Help

Enter the name of the entity you wish to map and then click the Create table button. Add each attribute to table but using the Add attribute button.

To add the attribute as a key or foreign key (or both) use the checkboxes below the input box, an attribute can be added as both a key and foreign key at the same time, and a key will appear underlined in the table.

If you need to delete an attribute use the delete link under the attribute you want to delete

If nothing needs to be done for this step or you have mapped every regular entity click the Check all tables button.

Feedback

1. The entity you choose to map must have the same name as an entity from the diagram.

2. For Step 1 of the algorithm, just choose to map the regular entities.

3. Choose an attribute of the entity to be a key for this relation.

Figure 3.5: The Help area on the left is replaced by the Feedback area on the right when a student requests feedback.

3.4 Problem and Solution Representations

ERM-Tutor is a problem solving environment for mapping entity relationship diagrams to relational database tables. The problems therefore are presented to the student as a graphical image of an entity relationship diagram. However, to generate ideal solutions to compare a student's solution we need a textual representation that can be parsed by the problem solver. So there are two categories of representations for each problem in the system: the representation to the student, and a textual representation used internally by the tutor.

The problems were taken from the ER-Tutor problem set [27]. The ER-Tutor is a tutoring system in which students learn to create an entity relationship diagram from a textual problem description. The ERM-tutor is teaching how to map the diagram which is the next step in database creation and so using ERM-Tutor would be a natural progression after using the ER-Tutor. Using the same problems means the student can see the progression of a database problem description through to relational database tables.

The problems are stored in a problems definition file that is loaded at the initialisation of the system. For each entry in the problem definition we store the problem number, the name of the image file associated with the problem, a textual description of the problem, and the internal representation of the problem. There are twenty seven problems in ERM-Tutor. However, new problems can easily be added to the system by altering the problem definition file and supplying a corresponding image of the Entity Relationship diagram.

3.4.1 Graphical Representation of Problems

ER-Tutor has a set of images of entity relationship diagrams as ideal solutions to be displayed to students when they ask to see the solution for a problem (Figure 3.6), and these images were used as the problem diagram for each problem in ERM-Tutor. Some of the images had to be altered as ERM-Tutor requires that key attributes are unique across the entire diagram, not just the particular entity.

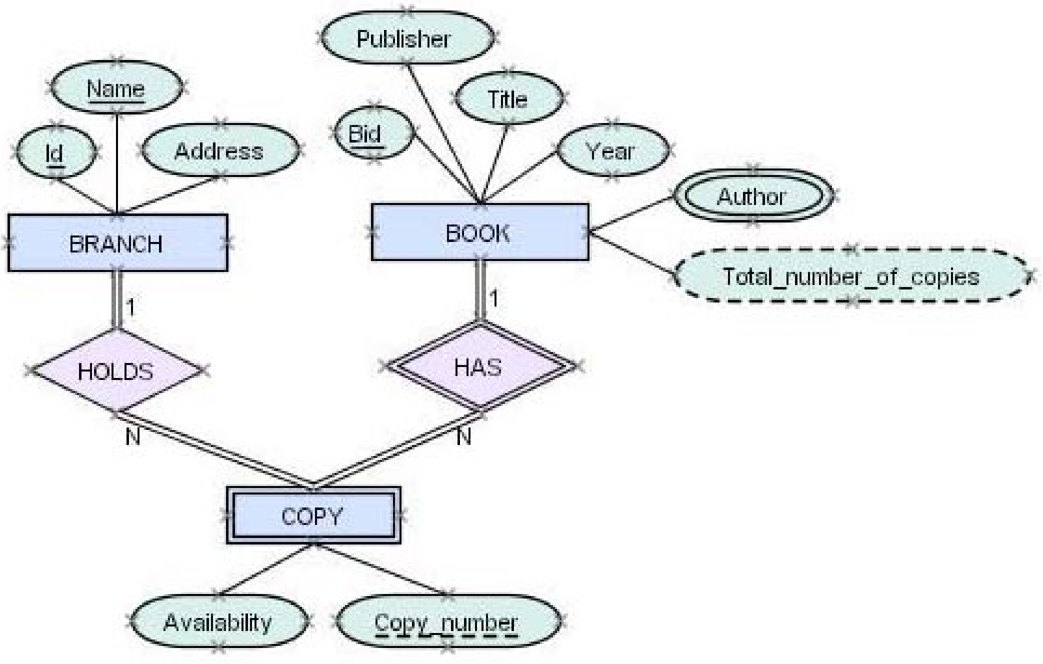


Figure 3.6: Graphical problem representation in ERM-Tutor.

There is also a textual description of each problem which is taken from the problem description in ER-Tutor. This is not needed by the student to solve the mapping problem, but was included as some extra information in case the student wanted clarification on any aspect of the entity relationship diagram by looking at the problem text.

3.4.2 Internal Representation of Problems

The internal representation of the problem is used by the constraints and the problem solver to evaluate the student's solution. The problem is stored as four lists: entities, relationships, attributes and connections. Each of the lists is made up of entries where the beginning of each entry is delimited by a "@". An entry in the entity list consists of a unique identifier for the entity, the entity name, and the entity type which can be "regular" or "weak". The relationship entry has a unique identifier for the relationship, the relationship name, and the relationship type which can be "regular" or "ident" for an identifying relationship. The attribute entry has a unique identifier for the attribute, the attribute name, the attribute type which can be "key" for a candidate key, "multi" for a multivalued attribute or "simple" for a simple attribute. The attribute entry then has a second attribute type for determining if the attribute is a composite ("composite"), or a component ("component"), or "simple" for a simple attribute, then lastly the entry has the identifier for the entity or relationship the attribute belongs to, (see Listing 3.1). The connections list is for describing the connections that join an entity to a relationship. The first element of the entry is the participation which can either be "total"or "partial", then the cardinality which is either "1" or "N" or "NC" if there is no cardinality specified. Next in the connections entry is the identifier for the first element in the connection, then the identifier for the next element in the connection. The internal representation of problems was taken from ER-Tutor [27].

```
("ENTITIES" "@ E1 TEXT_BOOK regular @ W1 CHAPTERS weak")

("RELATIONSHIPS" "@ I2 CONTAINS ident")

("ATTRIBUTES" "@ E1K1 ISBN key simple E1 @ W1S1 Total_no_of_references simple
 simple W1 @ W1K1 Chapter_No partial simple W1 @ W1S2 Topic simple simple W1
 @ W1S3 Total_no_of_pages simple simple W1 ")

("CONNECTIONS" "@ total 1 I2 E1 @ total N I2 W1 ")
```

Listing 3.1: Internal Representation of Solutions

3.4.3 Internal Representation of Solutions

The system has an internal representation of solutions, both the students' solutions and the solutions generated by the problem solver. The solution is made up of a series of named slots in the solution class that either contains a list or a string. These are *current-table*, a string name of the table being worked on, *current-atts* which contains a list of all the attributes in the solution (including the foreign keys and primary keys), *current-key* a list of the keys for the current table, *current-fkey*, a list of the attributes that are foreign keys, and *current-rel*, a string of the current relationship name.

There are some aspects of the solution that are only relevant to the student solution and not used in the ideal solution. One such aspect is the *mapped* slot which contains the list of names of relations that have already been correctly mapped by students at the step of the algorithm they are currently working on. There is also a list of the relations and their keys and attributes that have been mapped called *completed-tables*, this can be viewed by the student to help them when adding foreign keys. There is also a keys slot that is formatted the same way as the problem lists, again delimited by "a" which keeps track of the keys the student has chosen for each relation. The system needs to keep track of the keys chosen at early steps so it can determine if foreign keys chosen by the student are correct.

3.5 Problem Solver

Entity relationship to relational database mapping is a procedural algorithm. Because the task is procedural, it was possible to implement a problem solver to create the ideal solutions rather than having to write these manually for each problem. In the mapping algorithm often there is more than one option for a student to choose as a key for the relation, and this is a choice that follows through and affects future steps in the algorithm when specifying foreign keys. For this reason, it is important to have a problem solver in the system rather than relying on ideal solutions, which can not take into account earlier decisions of a student.

The problem solver comprises fourteen different functions, two for each step of the algorithm. The first function for a step is one that deals with the creation or alteration of a relation. It typically depends on the student having specified an element of the entity relationship diagram they are attempting to map. For example, in step one the student must specify which of the regular entities they are going to map and the problem solver will set the values of *current-key*, *current-fkey* and *current-atts* for the ideal solutions. This ideal solution that is created is based on the mappings already completed by the student that have been deemed to be correct.

The ideal solution is used to evaluate the student's solution for the mapping of a single element of the entity relationship diagram, for example just one regular entity. Solution evaluation is triggered by the student clicking the "Check table" button. It is actually only the *current-atts* part of the ideal solution that is used for evaluating a student's solution as the foreign keys and primary keys are evaluated through constraints directly from the problem. The solutions generated by the problem solver are also used to display to the student the ideal solution if they request this from the feedback level.

The second function that the problem solver has for a step determines all the elements from the entity relationship diagram that need to be mapped. For example, for step 6 the problem solver will identify every multivalued attribute that needs a relation created for it. This is used when evaluating a student's solution when they have selected the "Check all tables" button, when they think they have finished a particular step.

3.6 Knowledge Base

The knowledge base is a set of constraints. The constraints are loaded from a constraints file at the initialisation of the tutor, and stored as a list internally. The constraints are organised into groups based on the step of the algorithm they relate to and within these groups are a mixture of syntactic and semantic constraints.

The format of the constraints can be viewed in Listing 3.2.

The hint message is given to the student when the feedback level is set to "Hint", and the explanation is given for the feedback levels "Detailed hint" and "List all errors" (See Section 3.8 for a more detailed explanation of the feedback levels). Numbers are used for the constraint identifiers and where the first digit in the number corresponds to the step the constraint belongs to.

The constraints are written in the constraint language developed by Antonija Mitrovic and used in Normit [18] and SQL-Tutor [17]. There are two important functions used within constraints: `match` and `bind`. The match function is used to match a pattern to an input. The pattern can contain string literals, wild-cards which are preceded by a "?*" and varibles preceded by a "?". The match function will save the matched wild-cards and variables to a bindings list. The bind function saves one value to a variable into the bindings list.

Constraint 12 (List 3.3) is an example of a simple constraint. In the relevance condition for this constraint, we first check that the task the student is working on is step 1. Then the constraint checks that that value for current-table in the student solution (SS) is not null, then this is bound to the variable "?t" using the bind function. For this constraint to be satisfied, the match function must find in the entities list (taken straight from the problem decription) an entity that has as its name the value bound to "?t". As a side effect, the match value will also bind ?tt to the unique tag for that entity. For example, is the student

```
( unique constraint identifier

  "hint message"

  (relevance condition)

  (satisfaction condition)

  "explanation"

  "clause"

)
```

Listing 3.2: Constraint Format

has “text_book” as the value for current-table, and the entities list is as follows (“@” “E1” “TEXT_BOOK” “regular” “@” “W1” “CHAPTERS” “weak”) then the constraint will be satisfied.

It is important when writing constraints that the relevance conditions are set to the right level of detail. This can reduce the processing required when analysing the constraints, and also limit the number of error messages a student receives to a managable amount. For example, in step 1 almost all the constraints will fail if the student has not specified a table name from the problem. Rather than bombard the student with more than 10 errors, it is better that they just get one error, as remedying this error may fix them all. This can be achieved by adding the condition that the student has specified a correct table name to the relevance of each condition. This also ensures that the minimum amount of processing occurs when solutions are evaluated, as less constraints will be relevant, so there is no need to check their satisfaction conditions.

To model the domain of Entity Relationship mapping we wrote 111 constraints. The main constraint base was developed by examining the mapping algorithm in depth, and also through solving the example problems for the different steps. This constraint base was enhanced as extra constraints were added after testing the system by purposely trying erroneous solutions to check constraints existed to catch them.

3.6.1 Syntactic Constraints

A syntactic constraint is one that deals with the syntax of a student’s solution independently of the problem. The syntactic constraints in ERM-Tutor are very simple ones. The reason for this is the interface controls most of the syntax so there is not much room for errors to occur. So the syntactic constraints in ERM-Tutor are restricted to constraints such as checking that portions of a student’s solution are not null, and that when foreign keys are added, they are the primary keys of the relation they are from. For example, Constraint 61 is detailed in Listing 3.4 and this constraint from step 6 checks the student has specified the name of an table they want to map.

3.6.2 Semantic Constraints

Semantic constraints make up the majority of the constraints in ERM-Tutor. Semantic constraints are problem-specific and are concerned with the relationship between the student’s solution and the ideal solution. For example constraint 28 in Listing 3.5 checks that there are no differences in the set of *current-atts*

```
(12
 "Choose the name of an entity to map from the diagram."
 (and (equalp (current−task SS) 'step1)
      (not (null (current−table SS)))
      (bind ?t (current−table SS)))
 (match '(?∗d1 "@" ?tt ?t ?∗d2) (entities SS) bindings)
 "Step 1"
 "The entity you choose to map must have the same name as an
  entity from the diagram."
 )
```

Listing 3.3: Constraint 12: Example of a constraint

in the ideal and student solution.

On some occasions the semantic constraint calculates the ideal solutions itself, by matching the appropriate elements in the problem lists, and comparing these to the student's solution. An example of such a constraint is 211 which is listed in Listing 3.6. Constraint 211 is relevant when the student is working on step 2, they have specified a weak entity to map, they have specified the correct identifying relationship for the weak entity and they have specified a foreign key. The constraint is satisfied when the foreign key is an attribute from the owner entity. To check the satisfaction condition, firstly the constraint uses match to save the entity tag that the foreign key attribute belongs to in "?et". Then using the variable "?rt" that was matched to the relationship in the relevance condition it checks there is a connection from the relationship "?rt" to the entity "?et". However, this is not enough to ensure the student's solution is correct, as if the student specified an attribute from the initial weak entity this would be satisified, so there is the further check that the entity the foreign key is from is different to the initial entity.

3.7 Student Modeller

A student modeller is the component of an Intelligent Tutoring System that develops the student model. The student model is a central part of an intelligent tutoring system. It is the tutor's representation of what the tutor believes about the student's knowledge, a minimum requirement for a student model is tracking the student's performance on the tutor's material [5]. The student model is used in ERM-Tutor by the pedagogical module in providing feedback to the student on their solutions. There is the potential for the student model to have a more meaningful role by extending the tutor to have an open student model, and

```
(61
 "You must choose an attribute to map!"
 (equalp (current−task SS) 'step6)
 (not (null (current−table SS)))
 "Step 6"
 "Choose an attribute to map and create a table with that name."
)
```

Listing 3.4: Constraint 61: Example of a syntactic constraint

using the student model to provide problem selection.(See Chapter 5.)

ERM-Tutor uses constraint-based student modelling where the student's knowledge is represented as a list of constraints against the correct knowledge. The student model therefore, can be maintained as a list of constraints, and includes the student's past successes and failures with each constraint.

In ERM-Tutor, the task of the student modeller involves evaluating the student's solution against the constraint list and identifying which constraints are relevant and which of the relevant constraints are satisfied, and recording the result in the student model. This process occurs each time the student asks for feedback on their solution. The student modeller from the constraint-based data normalisation tutor Normit [18] was adopted for the ERM-Tutor.

The constraints for ERM-Tutor were written in the same constraint language as Normit [18]. To implement the computational simplicity of pattern matching for evaluating constraints, the constraints are represented in to the form of a RETE network. The Rete match algorithm is an efficient way of comparing a collection of patterns to create a collection of objects [9]. ERM-Tutor employs the code to initialise the RETE network and use the network to evaulate the constraints from Normit.

There are two types of student models maintained in ERM-Tutor by the student modeller, both as constraint lists. One is a long-term persistent student model that is loaded at the beginning of a student's session, and saved at the completion of each problem and on logging out of the system. This model is a type of overlay model (see Section 2.1.1) and includes the history of the student's success or failure with each constraint, and also includes the a list of problems the student has completed. For example, an entry in the long term student model for constraint 412.

```
(412 (1 1 0 1) 3/4)
```

In the example the student has useed the constrait 412 four times, and on three of those four occasions the constriant was satisfied.

The second type of student model is a short term model which only looks at the constraints with respect to the current problem state disregarding their history. From this model we can see which constraints were

```
(28
 "Check that you have specified the correct attributes for the table."
 (and (equalp (current−task SS) 'step2)
      (not (null (mapped IS)))
      (not (null (current−table SS)))
      (bind ?t (current−table SS))
      (match '(?∗d1 "@" ?tt ?t "weak" ?∗d2)(entities SS) bindings))
 (null (list−set−difference (current−atts IS)(current−atts SS)))
 "Step 2"
 "For this step add each simple attribute, and only the component attributes
  of a composite"
)
```

Listing 3.5: Constraint 61: Example of a semantic constraint

relevant to the student's current solution, as well as which of those constraints were violated. This is the student model that is used for generating feedback by the pedagogical module.

3.8 Pedagogical Module

The pedagogical module is the part of the system that does the teaching. In ERM-Tutor this involves using the student model to give the student appropriate feeedback messages. In ERM-Tutor the pedagogical module uses the constraints that have been violated to give the student feedback that matches the errors they have made. The pedagogical functions were adapted from Normit [18].

There are five levels of feedback available in ERM-Tutor, from the most generic, to the most specific they are: Simple Feedback, Hint, Detailed Hint, List All Errors and Solution. The pedagogical module starts the student at the simple feedback level the first time the student requests feedback from the tutor. Each subsequent time feedback is requested for the same solution the pedagogical module jumps the feedback up one level of specificity untill it reaches the List all errors level, which it will stay on for the remaining attempts on that problem for the student.

The student at any time can manually change the feedback level before they check their solution. This is the only way the student can get to the solution level of feedback. The pedagogical module will not skip them to the solution as we want to encourage students to come to the solution themself and not just copy the solution in. The feedback will only ever be shown when the student has the answer incorrect. Regardless of the feedback level selected, that level will only ever be shown if the student's solution was incorrect, if the student's answer is correct they will be shown the message "Well done! Your answer was correct.".

A description of each level of feedback is detailed below:

- *Simple Feedback* This is the most generic level of feedback, and simply tells the student when they have an error. If the student's solution is incorrect they will get the message "Almost there, you made X mistakes. Try again!", where X is the number of constraints the student's solution violated.
- *Hint* The hint level gives the student a clue on one action they could take to remedy their solution. The message they get is "Almost there - still a few errors though. <Hint>. Try again!" In this case <Hint> is the hint message component of a constraint. An example of a typical hint message is "Check you don't have any extra attributes!".
- *Detailed Hint* The detailed hint level gives the student a more in depth explanation of one aspect of their solution that they are having problems with. The message for this level is "Almost there - still a few errors though. <Explanation>. Try again!". Explanation is the explanation component of a constraint. The level of detail of an explanation would be similar to: "There are some extra attributes in your tables. Check that each

  attribute is an attribute of the entity.".
- *List all Errors* This gives the student a numbered list of every error they have made, corresponding to each constraint that was violated. Following the number of the error is the explanation component of the constraint.

The pedagogical module could have a more significant role in the tutor if it were extended to control problem selection using the student model (see Section 5), as is the case in SQL-Tutor. In this project problem selection could be based on which step of the algorithm the student needs practice on, as the constraints are already grouped by step this would involve only a change to identify which problems are relevant for each step.

```
(211
 "The key of the owner entity should be included in the relation."
 (and (equalp (current−task SS) 'step2)
      (not (null (mapped IS)))
      (not (null (current−table SS)))
      (bind ?t (current−table SS) bindings)
      (match '(?∗d1 "@" ?tt ?t "weak" ?∗d2)(entities SS) bindings)
      (not (null (current−rel SS)))
      (bind ?r (current−rel SS) bindings)
      (match '(?∗d3 "@" ?rt ?r "ident" ?∗d4)(relationships SS) bindings)
      (match '(?∗d5 "@" ?p1 ?c1 ?rt ?tt ?∗d6)(connections SS) bindings)
      (not (null (current−fkey SS)))
      (bind ?fk (first (current−fkey SS)) bindings))
 (and (match '(?∗d7 "@" ?at ?fk ?atype ?a ?et ?∗d8)(attributes SS) bindings)
      (match '(?∗d9 "@" ?p2 ?c2 ?rt ?et ?∗d10)(connections SS) bindings)
      (not (equalp ?et ?tt)))
 "Step 2"
 "The key of the owner entity must include a foreign key in the
  new relation."
)
```

Listing 3.6: Constraint 211: Example of a semantic constraint

4 Evaluation

A preliminary evalution was carried out of ERM-Tutor. The aim of this evaluation was to investigate whether the tutor helped the students learn, and to discover whether it would be viable to continue development of the ERM-Tutor. The evaluation was carried out in October 2004 at the University of Canterbury. Students enrolled in COSC226, the introductory database course, were invited to interact with the ERM-Tutor and provided with login names to do so. The tutor was available as a webpage on the internet so that they could access it from any web-enabled computer. The students had attended lectures on Entity Relationship Mapping earlier in the course, and had an exam, scheduled for the 27th of October, which included a mapping task.

The first time a student logged onto the system they were required to complete a short four question pre-test (see Appendix A.1). They were then free to use the system for as short or as long a time as they chose. Each time a student submitted a solution, their solution was recorded in a student log, as was their short-term student model, see Section 3.7. The students' results in the exam were used as a post-test for the system (see Appendix A.2).

4.1 Interaction with the system

A total of 34 students logged into ERM-Tutor. However, four students used the tutor for less than 1 minute and so their logs were excluded. The amount of time the students spent interacting varied from 3 minutes to 3 hours and 54 minutes, and the mean interaction time was 36 minutes. Within the group of students there was a mix of people logging on once, twice and three times. A summary of the interaction can be seen in Table 4.1.

	mean	s.d
Time spent on problem solving (minutes)	37:05	44:57
Number of logins	1.48	0.69
Number of attempted problems	4.34	3.80
Number of completed problems	0.93	2.36
Time spent per problem (minutes)	10:45	14:21

Table 4.1: Interaction details for the evaluation study for all students

The mean number of problems attempted was 4.34 and the mean number of problems completed was 0.93. To discover the reason for this difference between attempts and successfully completed problems the student logs were examined. For the mapping tutor to record a problem as completed the students complete all seven steps of the algorithm. However, there were many problems which had no further actions to be taken after the first few steps. The students aborted the problem after they had mapped all elements of the Entity Relationship diagram even though they had not completed all the steps.

4.2 Learning Domain Knowledge

In ERM-Tutor a unit of domain knowledge is represented as a constraint. Each time the student submits a solution, a list of the constraints that the student satisfied and violated is recorded. Learning curves using the constraint data can be generated by plotting the probability of a constraint violation against the number of submitted solutions, or the number of problems attempted. If the constraint represents an appropriate unit of knowledge, the learning should follow a smooth curve [2]. The learning curve should decrease in terms of constraints as the probability of a student violating a constraint should get smaller as they learn from the system, and violate less constraints.

To generate general learning curves for the ERM-Tutor for each participant the probability of violating an individual constraint for each problem attempted was calculated. Then to calculate an estimate of the probability of violating a given constraint for a given problem we averaged the individual probabilities across all constraints. These estimated probabilities were plotted against the number of problems attempted. The same process was applied to work out the estimated probability of a constraint violation for a number of solutions submitted.

4.3 Results

To investigate whether the students' performances improved when learning with ERM-Tutor, the pre-test and post-test results were compared. We calculated the means for students who used the system for greater than ten minutes. The pre-test mean was 55.43, and the post-test mean was 64.49, these can be seen in Table 4.2. The post-test mean was greater than the pre-test mean, which is a positive result. However, the difference is significant when we carried out a one-tailed paired T-test (t = -1.51, p = 0.05).

	mean	s.d
Pre-test score	55.43	32.82
Post-test score	64.49	21.65

Table 4.2: Pre-Test and Post-Test Results for Students who used ERM-Tutor.

The post-test results were also used to compare the students who used ERM-Tutor to the students who did not. The mean post-test result for students who used ERM-Tutor for more than ten minutes was 64.49, and the mean post-test result for the rest of the COSC226 class (including the students who used the ERM-Tutor for less than ten minutes) was 52.62. A one-tailed unpaired T-test on this showed that the difference was significant (t = 2.26, p = 0.05). These results are summarised in Table 4.3.

The learning curve was generated for the probability of a constraint violation for the number of problems attempted. Again, results were included only for the students who interacted with the tutor for more than ten minutes. These can be seen in Figure 4.1. The learning curve fits the data with an R^2 value of 0.601. If we examine the curve for the first six problems, the fit is very close with an R^2 value of 0.9145. This can be seen in Figure 4.2. The probability of violating a constraint during the first problem is approximately 28% and this descreases to about 12% by the sixth problem. We also generated the learning curve for the probability of a constraint violation for the number of attempted solutions submitted for students who used the tutor for more than ten minutes. This curve can be seen in Figure 4.3 and fits with an R^2 value of 0.623. The probability of a constraint violation began at about 23% and dropped to remain around 18%.

We generated the same learning curves a second time. This time we only used data from students who

	mean	s.d
Students who used ERM-Tutor	64.49	21.65
Student who did not use ERM-Tutor	52.62	25.39

Table 4.3: Post-Test Results for Students who used ERM-Tutor and those who did not.

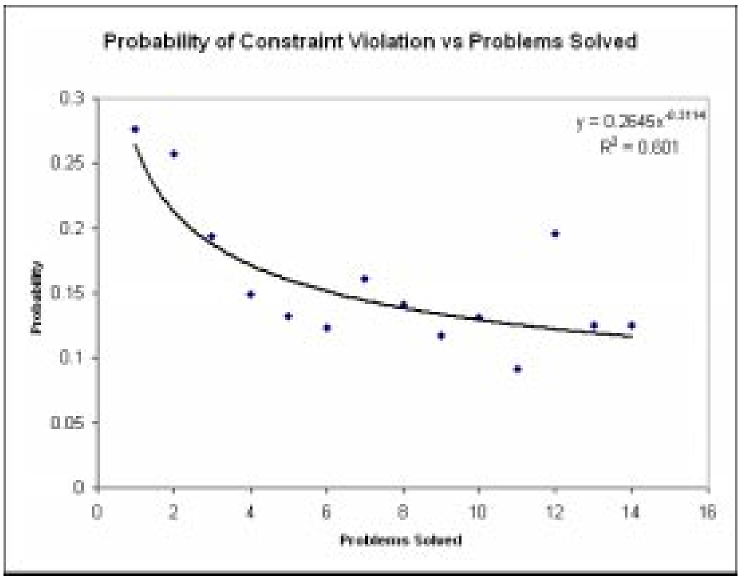


Figure 4.1: Probability of violating a constraint as a function of the number of problems the student attempted.

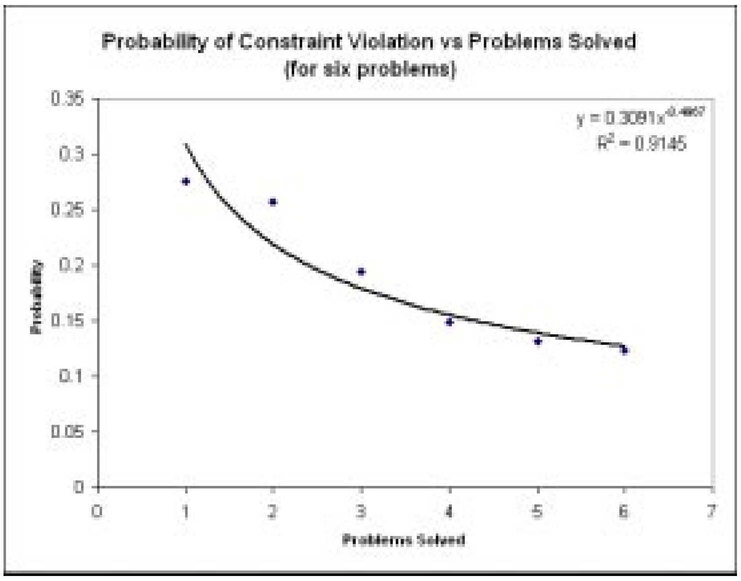


Figure 4.2: Probability of violating a constraint as a function of the number of problems the student attempted for the first six problems.

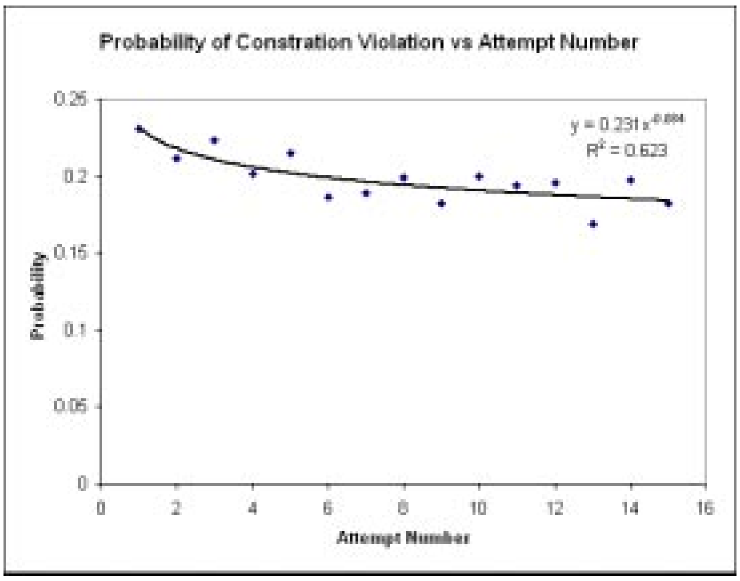


Figure 4.3: Probability of violating a constraint as a function of the number of attempted solution the students submitted.

used the ERM-Tutor for more than thirty minutes. Figure 4.4 shows the curve for all students over the whole set of problems solved. The curve does not fit the data point as well in this example with an R^2 value of 0.5679. Again the initial probability of violating a constraint starts at 28%, this value should match the value for the students who used the system for more than ten minutes, as the students should start at the same level. Again the curve for the first six problems only was generated, and can be seen in Figure 4.5. This shows that after six problems the probability of a constraint violation dropped to 11%. This curve fits the data points very well for this Graph, with an R^2 value of 0.9242. There was also a graph generated for the constraint violations as a function of the attempted solutions. This can be seen in Figure 4.6 and has an R^2 value of 0.6096, and here the probablity of a constraint violation starts at 23% and levels at 18%.

4.4 Discussion

4.4.1 Interaction with the System

The fact that many students did not completely finish problems in the system can not be interpreted as the students having missing domain knowledge. The likely reason for this is students stopped solving a problem once there were no more relevant steps. This could be remedied in ERM-Tutor by adding a new function where the student can specify if they believe there are no further steps to be completed for a particular problem. This would involve adding a button to the interface, and writing a new constraint to evaluation this and give the student feedback.

Overall it is a positive result that many students logged into ERM-Tutor more than once. It is also a positive that the average time students used the system was over half an hour, as the system was completely voluntary and students were being asked to volunteer at a busy time of the year while they were studying for exams. Both of these results suggest that students found the system useful and thought from their initial experience that ERM-Tutor would help them learn.

4.4.2 Pre and Post Test Results

Despite the fact that there was no significant difference between the pre and post test results, it is a favourable sign that the post test result was higher. There were some confounding factors that may have

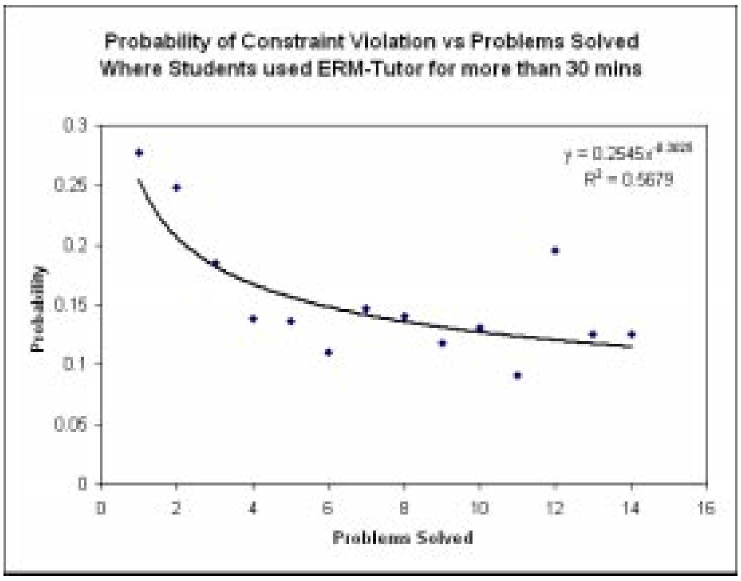


Figure 4.4: Probability of violating a constraint as a function of the number of problems the student attempted averaged over students who used ERM-Tutor for more than thirty minutes.

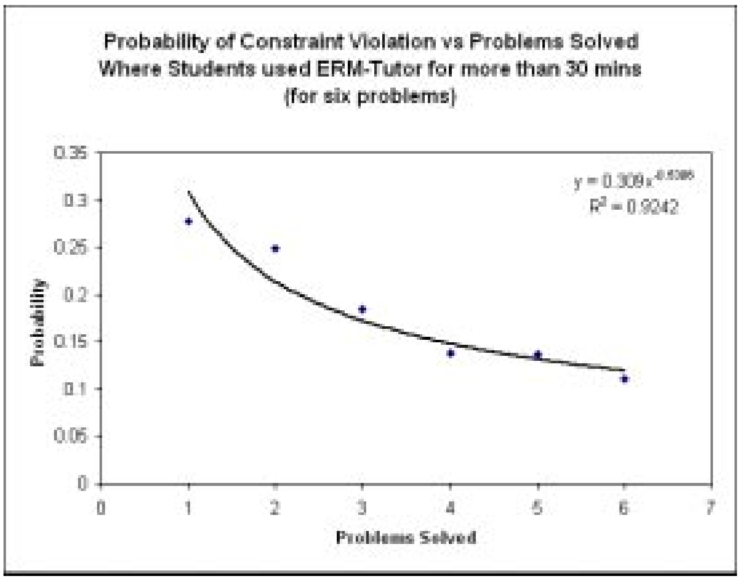


Figure 4.5: Probability of violating a constraint as a function of the number of problems the student attempted for the first six problems averaged over students who used ERM-Tutor for more than thirty minutes.

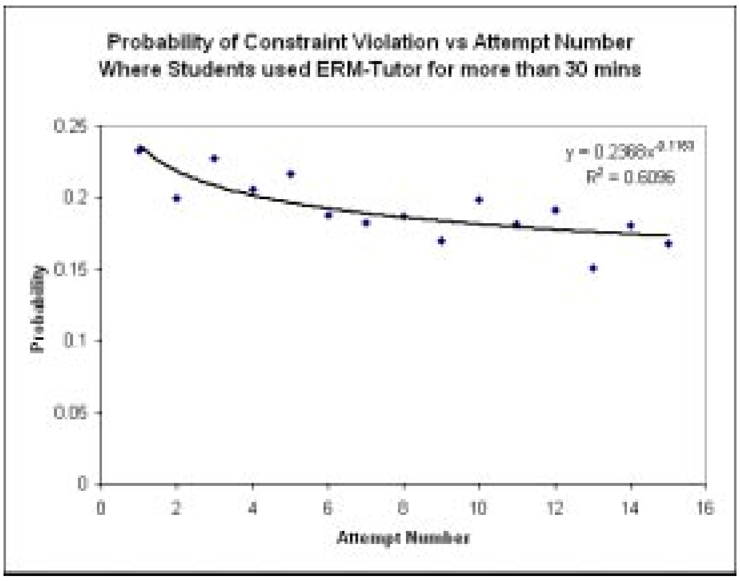


Figure 4.6: Probability of violating a constraint as a function of the number of attempted solution the students submitted averaged over students who used ERM-Tutor for more than thirty minutes.

contributed to the lack of a signifcant difference. The first factor is the small number of participants using the ERM-Tutor, this was due to participation in the system being completely voluntary with no reinforcer offered other than the chance to aid their study. A second confounding factor is that the nature of the two tests was quite different, with differing complexities. Because the study was so close to the exam there was no opportune time to offer a voluntary post-test, and many students only logged onto the system once. The exam questions on mapping in the exam that were used as the post-test was one question in which the students had to map an Entity Relationship diagram, whereas the pre-test only had four questions and was multichoice.

The result that the students who used ERM-Tutor had a statistically significantly higher mean exam result can only give an indication that the tutor may improve learning. Because there was no comparison carried out between the two groups prior to the post-test we can not conclude that it was ERM-Tutor that made the tutor users perform better. It is likely that some of the students who used ERM-Tutor were more motivated in their studies and are therefore students who would perform better regardless of the tutor. However, this result may suggest that ERM-Tutor contributed to these students' better performance, and gives a good indication that ERM-Tutor should be further evaluated.

4.4.3 Learning Domain Knowledge

The learning curves provide good support for the hypothesis that students can learn Entity Relationship mapping with ERM-Tutor. All of the learning curves showed a decreasing curve which means that the students' performances were improving. The drop in the probability of a constraint violation for all students against the the number of problems attempted was from 28% to 12% which is a good improvement made over the average student's use time of 37 minutes.

5 Conclusions and Further Work

Relational databases are extremely commonplace in today's modern society, and an important skill in designing these is mapping from a conceptual database schema to relational database tables. In many university database courses a mapping algorithm to translate from an Entity Relationship Diagram, to Relational Database Tables is taught, and this mapping skill is one in which students frequently make mistakes. One-to-one human tutoring is not feasible in such university environments. However, ITSs have been developed in many domains with the goal to individualise tuition as a human tutor would. We developed ERM-Tutor, an ITS developed to assist students learning the mapping algorithm.

ERM-Tutor is a problem solving environment where students can practise mapping Entity Relationship diagrams. The system breaks the problem up into the seven steps of the algorithm and the student is led through these sequentially. ERM-Tutor is a Constraint-Based Tutoring system. Constraint-Based Tutoring represents the student model as an overlay of domain knowledge, where the domain knowledge is represented with constraints. The ERM-Tutor is a web-based system enabling a number of students on a variety of platforms to use it simultaneously.

A preliminary evaluation study of ERM-Tutor was conducted in October 2004 at the University of Canterbury. Students enrolled in COSC226, an introductory database course, were invited to use the system as part of their preparation for the scheduled end of year examination. The preliminary study found promising results, as the students were using the system in their own time, and many chose to return to it on more than one occasion. The learning curves that were generated examining the probability of students violating constraints as a function of the number of problems they had worked on were decreasing curves. This showed that as the students were using the system they did learn, as they were less likely to violate constraints.

Formal evaluation is a vital part of the ITS development process to prove its credibility. The evaluation should be aimed to prove the effectiveness of the system in a classroom environment. This preliminary study that was carried out presented a positive case for ERM-Tutor's effectivenes. However, there needs to be a more in-depth study conducted to provide conclusive results that ERM-Tutor helps students learn. This study should involve an experimental group who will use the full ERM-Tutor system, and a control group for comparison. It would require a compulsory pre-test and post-test be conducted that were of similar complexity. Ideally it would have a greater number of participants, who were divided between the two groups, using the tutor for at least an hour.

There are a number of extensions that could be made to ERM-Tutor to increase its effectiveness. The student model that is maintained by the tutor could be opened up to the student by presenting it graphically in the tutor's interface. This would enable the student to see in which areas they have more mastery, and in which areas they may like to practise more. In an experiment investigating the effect of open student modelling in SQL-Tutor it was found that opening the student model boosted the performance of students [12] .

The pedagogical module of ERM-Tutor could have a more significant role if the tutor provided problem selection for the students. This would involve suggesting problems for the student that focus on areas they need to practise. In the mapping domain a suitable area may be a specific step of the algorithm. Many problems in the problem set only deal with two or three steps of the algorithm, so each problem could have a list of steps it is relevant to, and the pedagogical module could therefore chose a problem for the student that is relevant to the step in which they need practice. Problem selection has been successfully implemented in SQL-Tutor and has been shown to be effective in improving students' learning [20].

Another way to improve students' learning is to discourage shallow learning. Shallow knowledge is where students may guess the correct answer rather than applying domain theory. One approach to encouraging deep learning is to require the students to explain their solutions. In the Normit data normalisation tutor students who self-explained improved significantly in problem-solving, but this was not compared to students who did not self-explain [19]. Self-explanation could be added to ERM-Tutor to ensure that shallow learning does not occur.

Acknowledgments

I would like to thank my supervisor Antonija Mitrovic for the invaluable advice, and guidance. I'd also like to thank past and present members of the Intelligent Computer Tutoring Group for supplying some of the code used in the ERM-Tutor system. Thank you also to Pramudi Suraweera for developing Kermit which provided the internal representations used in ERM-Tutor, as well as the problem set, and served as a guide for developing this tutor. Thank you to the students in the evaluation study who gave up their time to use ERM-Tutor. Finally thank you to Konstantin Zakharov for offering his opinion on numerous occcasions during the development of ERM-Tutor.

Bibliography

[1] J. Anderson and B. Reiser. The lisp tutor. In *Byte*, 1985.

[2] J. R. Anderson. *Rules of the Mind*. Lawrence Erlbaum Associates, 1993.

[3] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier. Cognitive tutors: Lessons learned. In *Journal of Learning Sciences*, 1996.

[4] T. A. Atolagbe. Simtutor: a multimedia intelligent tutoring system for simulation modeling. In *Proceedings of the 29th conference on Winter simulation*, 1997.

[5] J. Beck, M. Stern, and E. Haugsjaa. Applications of AI in Education. `http://www.acm.org/crossroads/xrds3-1/aied.html`, 1996.

[6] B. Bloom. The search for methods of group instruction as effective as one-to-one tutoring. In *Educational Researcher*, 1984.

[7] P. Chen. The entity relationship model - toward a unified view of data. In *ACM Transactions Database Systems*, 1976.

[8] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 3rd edition, 2000.

[9] C. Forgy. Rete: A fast algorithm for the many pattern/¡any object pattern matching problem. In *AI*, 1982.

[10] Franz Inc. Franz Inc: Allegro Common Lisp and Common Lisp Products. `http://www.franz.com/`, 2004.

[11] R. Freedman. Atlas: A plan manager for mixed-initiative, multimodal dialogue. In *AAAI-99 Workshop on Mixed-Initiative Intelligence*, 1999.

[12] D. Hartley and A. Mitrovic. Supporting learning by opening the student model. In S. Cerri, G. Gouarderes, and F. Paraguacu, editors, *Proc. 6th International Conference on Intelligent Tutoring Systems ITS 2002*, 2002.

[13] P. Holt, S. Dubs, M. Jones, and J. Greer. The state of student modelling. In *Student Modelling: The Key to Individualized Knowledge-Based Instruction*, 1994.

[14] A. Lesgold, S. P. Laijoie, M. Bunzo, and G. A. Eggan. A coached practice environment for an electronics troubleshooting job. In *Proc. of Computer Assited Instruction and Intelligent Tutoring Systems: Establishing Communication and Collaboration*, 1990.

[15] B. Martin and A. Mitrovic. Its domain modelling: Art or science? In *Artificial Intelligence in Education*, 2003.

[16] B. I. Martin. *Intelligent Tutoring Systems: The Practical Implementation of Constraint-based Modelling*. PhD thesis, University of Canterbury, 2001.

[17] A. Mitrovic. A knowledge-based teaching system for sql. In *Proc. ED-Media/ED-Telecom ‘98*, 1998.

[18] A. Mitrovic. Normit, a web-enabled tutor for database normalization. In R. Kinshuk, R. Lewis, R. Akahori, T. Kemp, T. Okamoto, L. Henderson, and C.-H. Lee, editors, *Proc. ICCE 2002*, 2002.

[19] A. Mitrovic. Supporting self-explanation in a data normalization tutor. In V. Aleven, U. Hopppe, J. Kay, R. Mizoguchi, H. Pain, R. Verdejo, and K. Yacef, editors, *Supplementary Proceedings, AIED 2003*, 2003.

[20] A. Mitrovic and B. Martin. Scaffolding and fading problem selection in sql-tutor. In V. Aleven, U. Hopppe, J. Kay, R. Mizoguchi, H. Pain, R. Verdejo, and K. Yacef, editors, *Supplementary Proceedings, AIED 2003*, 2003.

[21] A. Mitrovic, M. Mayo, P. Suraweera, and B. Martin. Constraint-based tutors: a success story. In L. Monostori, J. Vancza, and M. Ali, editors, *Proc. 14th Int. Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE-2001*, 2001.

[22] A. Mitrovic and S. Ohlsson. Evaluation of a constraint-based tutor for a database language. In *International Journal on AIED*, 1999.

[23] S. Ohlsson. Constraint-based student modeling. In *Student Modelling: The Key to Individualized Knowledges-Based Instruction*, 1994.

[24] S. Ohlsson. Learning and Instruction: An Introduction. COSC420 Course Reading Material, July 2004.

[25] J. A. Self. Bypassing the Intractable Problem of Student Modeling. In C. Frasson and G. Gauthier, editors, *Intelligent Tutoring Systems: at the Crossroad of Artificial Intelligence and Aducation*, pages 107–123. Ablex Publishing Corporation, Norwood, NJ, 1990.

[26] P. Suraweera. An animated pedagogical agent for sql-tutor. Technical report, University of Canterbury, 1999.

[27] P. Suraweera. An intelligent teaching system for database modelling. Master's thesis, University of Canterbury, 2001.

[28] R. B. Wilmot. Foreign keys decrease adaptability of database designs. *Commun. ACM*, 27(12):1237–1243, Dec. 1984.

Appendix A

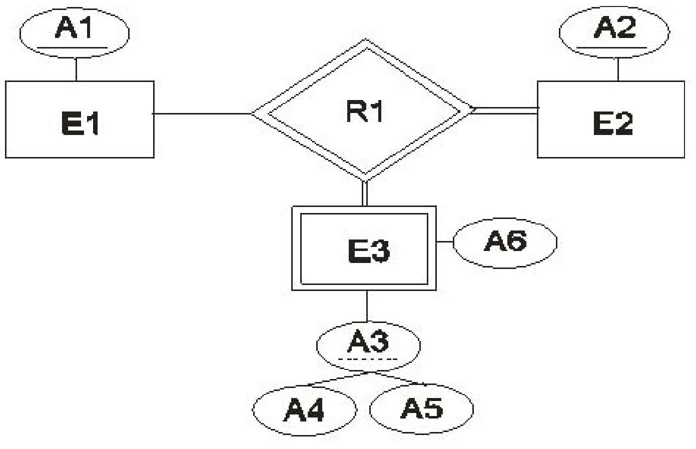
Appendix

A.1 pre-test

1. Which of the following options is true for mapping a multivalued attribute that belongs to a relationship type?
   (a) A foreign key is added to the relation corresponding to the entity type that participates fully in the relationship type.
   (b) There is a new relation created for the multivalued attribute.
   (c) A foreign key is added to the relation corresponding to the entity type on the N side of the relationship type.
   (d) There is a new relation created for the relationship type, which also contains the mutlivalued attribute.
2. When mapping an ER schema into a relational schema, in what situation a new relation is NOT added to the database:
   (a) for each M:N binary relationship type
   (b) for each higher degree relationship
   (c) for each multivalued attribute
   (d) for each weak or regular entity type
   (e) for each 1:1 or 1:N binary relationship type
3. When mapping a binary 1:N relationship type, a foreign key is added to:
   (a) the relation corresponding to the entity type on the 1 side of relationship type
   (b) the relation corresponding to the entity type on the N side of relationship type
   (c) both relations
4. Table R3 corresponds to the weak entity type E3 from the given ER diagram (see Figure A.1).
   (a) True
   (b) False

A.2 post-test

See Figure A.2 for the post-test.



R3(A1, A2, A3, A4, A5, A6)

Figure A.1: Pre-test diagram.

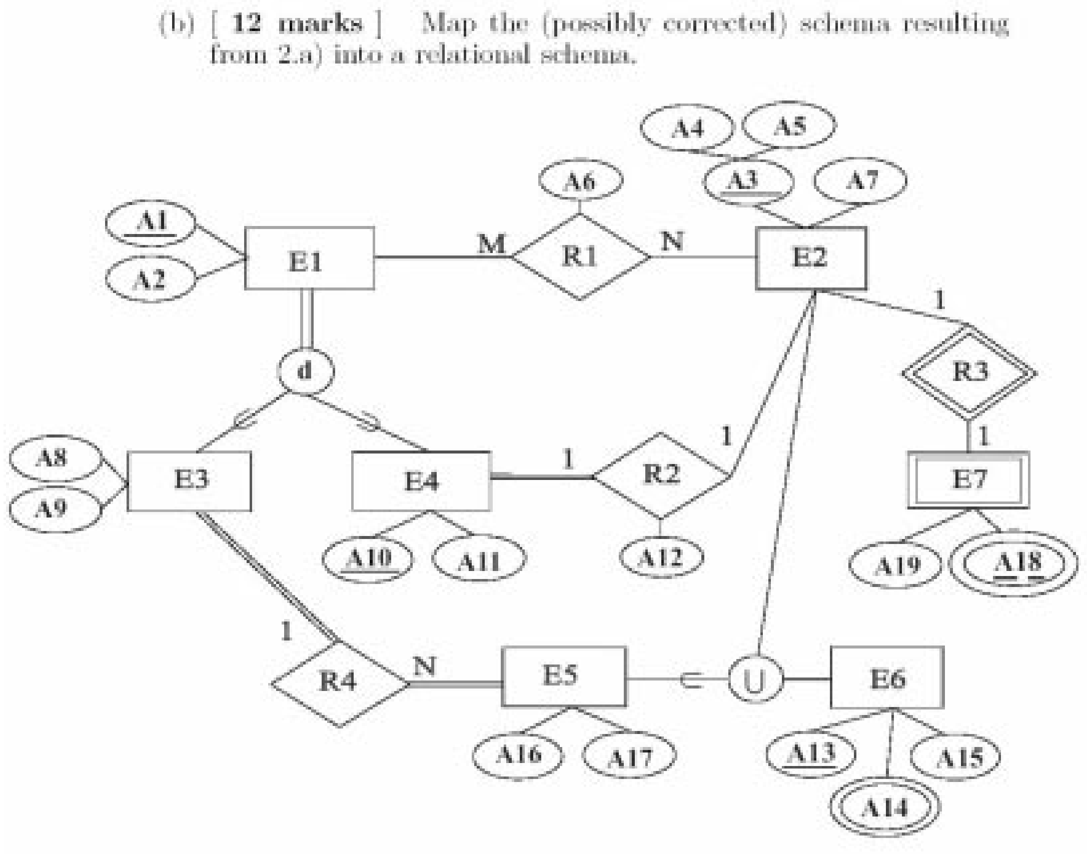

(b) [**12 marks**] Map the (possibly corrected) schema resulting from 2.a) into a relational schema.

Figure A.2: Post-test.