Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

TOWARDS DESIGNING A KNOWLEDGE-BASED TUTORING SYSTEM: SQL–TUTOR AS AN EXAMPLE

by
Gang Zhou

A Knowledge-Based Tutoring System, also sometimes called an Intelligent Tutoring System, is a computer based instructional system that uses artificial intelligence techniques to help people learn some subject. The goal of the system is to provide private tutoring to its students based on their different backgrounds, requests, and interests. The system knows what subject materials it should teach, when and how to teach them, and can diagnose the mistakes made by the students and help them correct the mistakes.

The major objective of this dissertation is to investigate and develop a generic framework upon which we can build a Knowledge-Based Tutoring System effectively. As an example, we have focused on developing SQL–TUTOR, a tutoring system for teaching SQL concepts and programming skills. The generic architecture of the system is rooted at the popular view that a tutoring process between a tutor (either a human being or a machine) and a student is a knowledge communication process. This process can be divided into a series of communication cycles and each communication cycle consists of four phases, namely, planning, discussing, evaluating, and remedying phases.

One major feature of the architecture proposed by us in this dissertation is its curriculum knowledge base which contains the knowledge about the course curriculum. We have developed a representation schema for describing the goal structure of the course, the prerequisite relationships among the course materials, and the multiple views to organize these materials. The inclusion of the curriculum

knowledge in a KBTS allows the system to create different curricula for each individual student and to diagnose the student's errors more effectively.

The system also provides a group of operators for the student to hand-tailor his/her curricula when he/she starts learning the course. The student can use these operators to select a specific path to go through the course materials, to pick a specific topic from the curricula to study, or to remove a particular topic from the curricula. Since the student can construct his/her own learning plans by these operators, he/she is relatively free to determine how to study the course materials and, as a result, he/she can become more active in the tutoring process.

The knowledge about a subject domain is stored in a set of topics and a sample database. The content of a topic consists of a set of related domain concepts. Each concept is described by both natural and formal forms. The relationships among the concepts are modeled a type of semantic network called the context network. The sample database contains a set of sample tables and an enhanced system catalog which contains the knowledge about the name, semantic meanings of the database objects. The built-in Problem Solver of the system allows the system to reason over the networks and the sample database and answer various kinds of questions raised by the student about the domain concepts and their relationships.

The knowledge of writing SQL queries is embodied in a set of examples attached to the topics. Each of such an example is carefully designed for one category of SQL query problems. An example in SQL–TUTOR is a packed knowledge chunk which can serve several important teaching purposes, including generating problem descriptions with different levels of details, formulating various SQL solutions for the given problem, explaining these solutions to the student, and evaluating SQL queries written by the student.

TOWARDS DESIGNING A KNOWLEDGE-BASED TUTORING SYSTEM: SQL–TUTOR AS AN EXAMPLE

by
Gang Zhou

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Computer and Information Science

May 1996

APPROVAL PAGE

TOWARDS DESIGNING A KNOWLEDGE-BASED TUTORING SYSTEM SQL–TUTOR AS AN EXAMPLE

Gang Zhou

Dr. Peter A. Ng, Dissertation Advisor — Date
Professor of computer and Information Science Department, NJIT

Dr. James McHugh, Committee Member — Date
Full Professor of Computer and Information Science Department, NJIT

Dr. Richard Scherl, Committee Member — Date
Assistant Professor of Computer and Information Science Department, NJIT

Dr. Qian-Hong Liu, Committee Member — Date
Assistant Professor of Computer and Information Science Department, NJIT

Dr. Alexander Pasik, Committee Member — Date
Member of Gartner Group, Stamford, CT

Raymond T. Yeh, Committee Member — Date
Distinguished Professor of Computer and Information Science Department, NJIT

BIOGRAPHICAL SKETCH

Author: Gang Zhou

Degree: Doctor of Philosophy

Date: May 1996

Undergraduate and Graduate Education:

- Master of Computer and Information Science,
  Jilin University, Changchun, China, 1984
- Bachelor of Computer Engineering,
  Jilin University, Changchun, China, 1981

Major: Computer and Information Science

Presentations and Publications:

G. Zhou, Curriculum Knowledge Representation in SQL–TUTOR, in *Proceeding of ED-MEDIA 94–World Conference on Educational Multimedia and Hypermedia*, Vancouver, British Columbia, Canada, 1994.

G. Zhou, J. T. L. Wang, and P. A. Ng, 'A Knowledge-Based Tutoring System for SQL Programming, in *Proceedings of 6th IEEE International Conference on Tools with Artificial Intelligence*, New Orleans, Louisiana, 1994.

G. Zhou, J. T. L. Wang, and P. A. Ng, Curriculum Knowledge Representation and Manipulation in Knowledge-Based Tutoring System, to appear in *IEEE Transactions on Data and Knowledge Engineering*, 1996.

To my parents, for their devotion, to my daughter, Amy, who loves to draw pictures on this thesis, to my son, Wilson, who sometimes spends more time on his homework than his father, and to my wife, Lucy, for her love, understanding and support.

ACKNOWLEDGMENT

It is my pleasure to acknowledge the advice and assistance from my advisor and colleagues who encouraged me while I wrote this dissertation. My special appreciation goes to Dr. Peter A. Ng, my advisor, for his insightful comments on and constructive criticism of the dissertation, without which I could never have done this work.

TABLE OF CONTENTS

Chapter **Page**

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this chapter, an overview is presented for the research in the area of Knowledge-Based Tutoring System (sometimes also called Intelligent Tutoring Systems), a branch of computer applications in education. We first describe the goals of the research in this field by briefly reviewing its history; then specify the most important issues underlying the development of such systems; and finally discuss some major weaknesses found in the existing systems, from which the research described by this dissertation is motivated.

1.1 Development of Knowledge-Based Tutoring System

The applications of computer technology in education have been under development since the early 1960s [60]. A variety of computer-assistant instructional systems were developed [26, 34, 35, 36, 49, 71, 72] even in the early age of computer. For historical reasons, the early research of computer on education has been conducted under the name *Computer-Assistant Instruction (CAI)*.

Traditional CAI has its root in the goal of building interactive teaching devices. One of the most influential efforts in CAI was the work of Suppes and his associates [71]. The major contribution of their work was adding brief drill and practice sessions to regular instruction. This significantly improved the student's achievement in basic skill areas. Another influential CAI system in the 1960s was TICCIT (Time-Shared, Interactive, Computer-Controlled Information Television) [11, 43]. The main goal of TICCIT was to design a cost-effective instructional delivery system. TICCIT was tested at many schools and eventually became a commercial system.

These early systems opened a new era of CAI. They provided an opportunity for a large number of people working in the area of computer education to gain

practical experience with CAI and resulted in the development of many more systems. One important achievement by these later CAI developments was made towards the architecture of a system. In the early forms of CAI, all the components (such as subject materials, student information, instructional strategies) were combined and stored in the same file. In the early 1970s, Seidel and his associates developed a prototyped computer training system for the Army Personnel [56, 67] in which the subject materials and instructional decision rules are separated in different data files. As the result of this separation, it becomes easy and simple to modify any instructional rules without requiring any reformulation of the whole system.

In spite of the widespread use of CAI in various educational applications, early CAI systems have the limitation that all the instructional processes, including presentation formats, as well as the interactions between the systems and its users, have to be specified in the program at the time when the systems are constructed. As the result of this limitation, the systems provide the same tutoring to all the students, and ignore the differences in their backgrounds, achievements, and learning attitudes. In other words, these systems can only provide very restricted specialized instruction to an individual student.

Knowledge-Based Tutoring Systems (KBTS) (sometimes also called Intelligent Tutoring Systems (ITS)) are computer-based instructional systems developed more recently [1, 9, 17, 22, 31, 42, 48, 54, 55, 59, 60, 65, 70, 77, 78, 81]. The major difference between a KBTS and a traditional CAI is that the KBTS uses artificial intelligence techniques to help a person learn the intended subjects. In 1970, Carbonell developed SCHOLAR system for teaching simple facts about South American geography [17]. The system organizes the subject contents by a complex and well-defined semantic network, in which each node represents a geographical object and the links between the nodes represent the geographical relationships. It also has other prominent

features such as the Socratic style of teaching and the domain independent inference strategies.

SCHOLAR was extended to the WHY system by Carbonell's colleagues [68, 69]. The WHY system tutors students about the causes of rainfall, which is a very complex geographical process involving many factors. In this system, the subject contents are stored in a hierarchy of scripts that represent the stereotypical sequences of rainfall events. The major contribution of this project is the formulation of a set of tutorial rules which implement the Socratic style of teaching.

The SOPHIE (a SOPHisticated Instructional Environment) system [9] can provide its students a "reactive learning environment", in which the students can learn how to diagnose deficient components in an electrical circuit by trying out their own ideas rather than just receiving instructions from the system. It incorporated a qualitative model of electrical circuits and a rule base for answering students' questions and simulating human reasoning.

The idea of the reactive learning environment in SOPHIE have been applied to BUGGY [8] and QUEST (Qualitative Understanding of Electrical System Troubleshooting) [77, 78] for constructing diagnostic models. The purpose of BUGGY is to teach students to learn basic mathematical problem-solving skills. It provides a mechanism for explaining why a student is making a mistake, instead of only identifying the mistake. Both QUEST and SOPHIE use a causal calculus for its internal representation of an electrical system that is directly influenced by qualitative reasoning, and both of them have the same application domain and adopt similar tutorial strategies. However, the graphic simulations and causal explanations of circuit behavior play an important role in QUEST.

WEST [14] was developed for investigating diagnostic strategies required to explain students' misunderstanding from their observed behaviors. The subject chosen for the tutoring is a computer educational game called *How the WEST*

was won, which involves the applications of various arithmetic skills. WEST adopted *coaching*, a new tutorial strategy, to teach the appropriate manipulations of arithmetic expressions. The goal of the coaching strategy is to have the students to enjoy the game and learn the subject as a consequence of fun.

GUIDON [20, 21] is a program for teaching medical diagnostic skills. Using the rules of the MYCIN (which is an expert system for medical diagnoses) as its tutoring materials, GUIDON teaches the students about the relevant clinical and laboratory data and how to use that information for diagnosing the causative organism. It differs from other KBTSs in terms of the dialogue form between the system and a student. In order to engage the students in a dialogue about a patient's suspected infection, GUIDON applied the *mixed-initiative* style of dialogue which can be controlled either by the system or the student.

Anderson and his associates developed computer tutoring systems for teaching high school geometry [1] and LISP programming [2, 54]. The systems were designed according to a set of pedagogical principles derived from Anderson's ACT* learning theory [1]. Each of their systems has three models:

- Ideal Student Model: representing the necessary knowledge to solve the various domain problems;
- Bug Catalogue: representing the knowledge about the common mistakes and poor strategies of a novice programmer; and
- Tutoring Control Module: representing the pedagogical strategies applied by a system to communicate with the student.

The Model-Tracing Methodology is used to match each of the student problem solving behaviors to the rules in the Ideal Student Model or the Bug Catalog. If a match is found from the Ideal Student Model, then the student is on the right track. If a match is found from the Bug Catalog, then the student is on the wrong track.

One of the major concerns of [32, 39, 64] is to follow a consistent style to teach students programming. From interviews with tutors and videotapes of interactive tutoring sessions, they identified five main issues, called *tutorial considerations*, which have influence on the tutor to make decisions about which bugs (errors) to be tutored, when to tutor these bugs, and how to tutor them. The five tutorial considerations are:

- how critical a bug is;
- what category a bug is;
- what cause a bug to occur;
- what are the appropriate tutorial goals for tutoring a bug; and
- what tutorial interventions would achieve the tutorial goals.

Although different KBTSs have taken different implementations, they have shared some common components in their system structures. In the following section, we will discuss the major components found in most of the existing KBTSs and the problems associated with their designs.

1.2 Major Components in Knowledge-Based Tutoring System

As pointed out in [12, 59, 60, 75, 76], a fully implemented KBTS should have the following three components (Figure 1.1):

- Domain Knowledge Base: storing the subject materials and skills that a student should learn from the course;
- Student Model: reflecting a student's background and performance in the course;

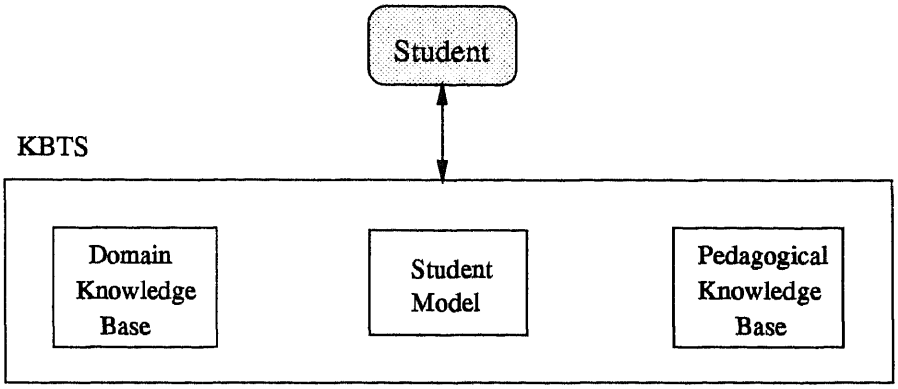


Figure 1.1 Three Major System Components for KBTS

- Pedagogical Knowledge Base: describing the tutoring strategies of the system.

1.2.1 Domain Knowledge

For each KBTS, there is a target domain of the subject materials that the system intends to teach its students. The target domain (or simply domain) can be any subject, such as mathematics, physics, chemistry, geography, programming languages, and English. In general, the domain knowledge can be divided into two categories: declarative knowledge and procedural knowledge. Declarative knowledge includes concepts in the domain and the relationships among the domain concepts. Procedural knowledge includes the rules and procedures for solving problems within a domain.

In a KBTS, the domain knowledge is contained in the *Domain Knowledge Base (DKB)*. Declarative knowledge is typically represented by semantic network [9, 10, 17], scripts-frame [24, 80, 81], and other knowledge representation schemata in which concepts are recorded in nodes and relations among the concepts are defined by links. As an example, the domain knowledge in SCHOLAR [17] is represented in a semantic network shown by Figure 1.2. Corresponding to a geographical object such as continent, country, and state, there is a node in the network composed of an object name and a list of the attribute-value pairs. For instance, the object `Argentina` has

the attributes `location, latitude, neighborhood countries`, etc.. Among these attributes, there is the part-of relationship between geographic objects, which allows the system to respond any questions of the form "Can `X` be a part of `Y`?"

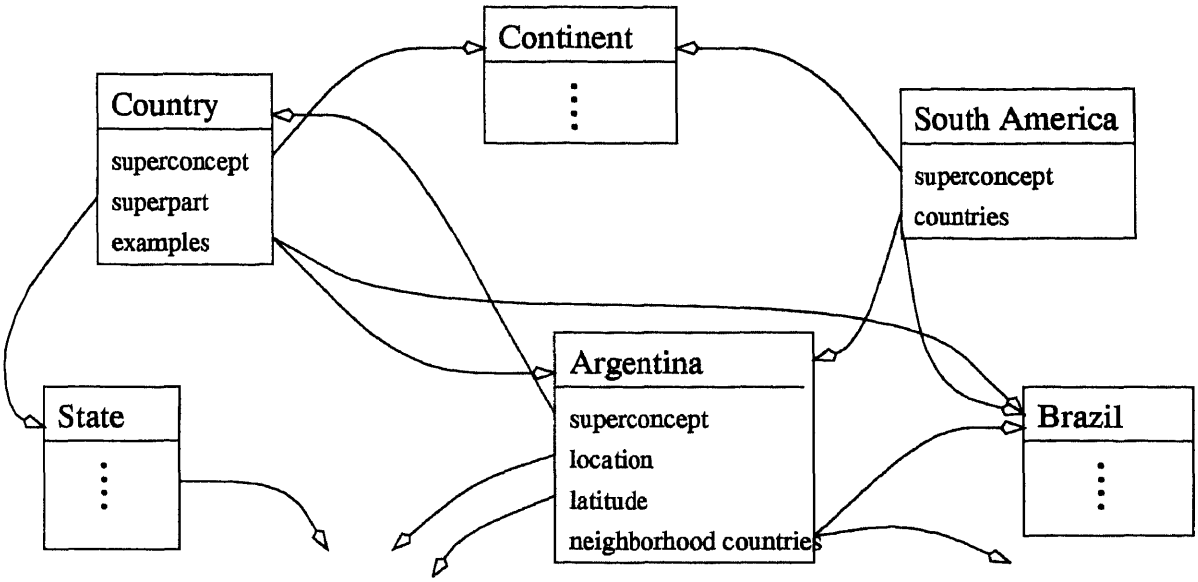


Figure 1.2 A Portion of SCHOLAR's Semantic Net

Procedural knowledge has traditionally been included in KBTSs that carry out procedural tasks, such as programs for solving arithmetic problems [8], or for simulating the operations of a stream engine or a recovery boiler [28, 79]. The typical representation schemata used for specifying the procedural knowledge by the KBTSs are qualitative model [77, 78], procedural representation [9], production systems [1, 14, 20, 21, 22, 54], and logic programs [42]. For example, the following LISP codes define an important rule for a system in equilibrium [80], which can be explained as follows: if the negative and positive x-force and y-force components of an object are equal, the object is in equilibrium.

```
(defrule in-equilibrium
   (is-object obj
      (and (equal (x-neg-force obj) (x-pos-force obj))
```

```
        (equal (y-neg-force obj) (y-pos-force obj))
(assert (in-equilibrium obj)))))
```

The domain knowledge base of a KBTS is a conceptualization of the domain subjects constructed by a teaching expert who designs the curriculum for the course. For this reason, a DKB is also called an expertise module [23, 50], since it represents the instructor's expertise about the domain. One problem associated with the design of the DKB is its completeness and soundness. A DKB is complete if it has encoded all the knowledge that the students should learn from the course; it is sound if it has encoded the knowledge correctly. If the instructor's knowledge about the domain is incomplete and incorrect, then the domain knowledge base can also be incomplete and incorrect. This tells us that the quality of a DKB depends on the knowledge quality of the instructor. How to develop a DKB with high quality for a KBTS is a very important, yet unstressed issue, but its discussion will beyond our interest here.

1.2.2 Student Model

In the process of studying a domain subject, a student will gradually construct and update a model in his[1] mind about the domain subjects. This model is an abstraction of his understanding about the domain and we call it a student mental model. Generally speaking, the initial student model is small and simple because the student knows very little about the domain subject at the very beginning of his study. As the learning continues, the student understands more concepts and masters more skills to solve various problems in the domain. As a consequence of this process, the student mental model grows both in its size and complexity.

At any stage of his learning, the student has both correct and incorrect knowledge about the subject, and has not learned some subject materials. Therefore,

[1]In writing a work such as this, it is inevitable problem to decide whether to use 'he' and 'his' or 'she' and 'her', where no implication with respect to gender is intended. Accordingly, we will simply use 'he' and 'his' in these cases.

some parts of his mental model reflect the correct conceptualization, whereas the other parts of his mental model are the wrong conceptualization about the domain knowledge, and some domain knowledge is missing from the mental model. Two terms, *missing conception* and *misconception*, have been used to describe a student's mastery of a subject. A student has a missing conception if he has not mastered an item of domain knowledge (e.g. a concept or a skill). A student has a misconception if he has some wrong knowledge about the subject.

It is a fundamental requirement for an instructor to evaluate a student's learning performance during a tutoring session, because no tutoring can be effectively accomplished without understanding what are known (both correct and wrong) and what are not known by the student. An experienced human instructor always tries to capture the mental model of a student about the subject domain and uses this type of knowledge to guide his teaching. This is also a major task of a KBTS. For this reason, many of the existing KBTSs maintain an internal module for realizing the abstraction of the student domain knowledge. Such a module is called a *student model (SM)*. Once such a model has been created in a tutoring system, the system can use it to find out the student's missing conceptions and misconceptions.

A student model may contain three kinds of information about the student domain knowledge: i) his correct knowledge; ii) his missing conceptions; and iii) his misconceptions. The student models in some KBTSs [10, 15, 17, 30] can represent only the missing conceptions from the students domain knowledge. This type of student model is called the *overlay student model.* Another common approach for student modeling is to create a *bug model* [13, 61] for the purpose of identifying both missing conceptions and misconceptions from a student.

Conceptually, an overlay student model is a proper subset of the domain knowledge base. The most prominent advantage of an overlay student model is that it can be implemented efficiently by attaching a binary number to each item

in the domain knowledge base. The values of these binary digits indicate whether the student has mastered this piece of knowledge or not. In Figure 1.3 (a), part D2 represents the complete set of domain subjects, and part D1, which is a subset of D2, represents the part of the domain subjects mastered by the student. In an overlay student model, all the items in D1 will be attached by the binary value 1, indicating this is the part of the overlapping between the expertise and student domain knowledge, whereas all the items not in D1 will be attached by the binary value 0, indicating that these items are missing from the student's knowledge.

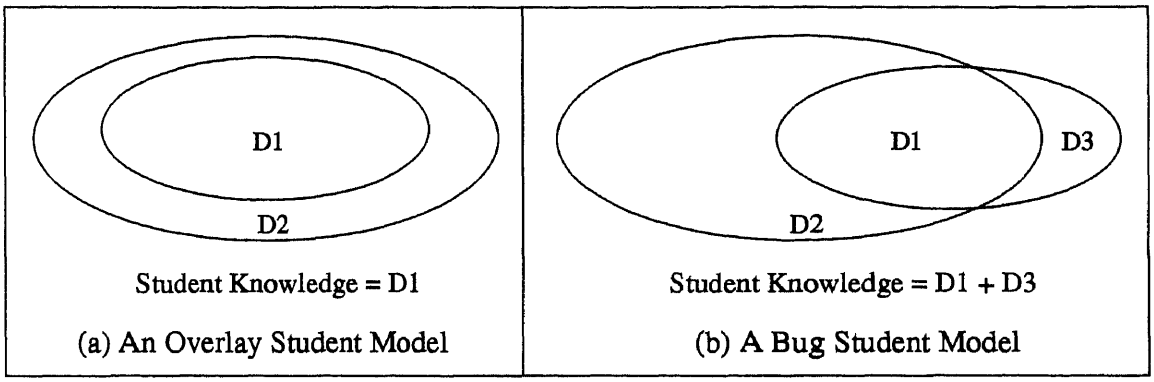


Figure 1.3 The Relationship between Domain Knowledge Base and Student Model

This type of simple overlay model can be improved by replacing the binary value with a variant which is a numerical value within a given range, thus it can be used to indicate more than two mastery levels of the student on the domain subjects. For example, a system can use 1 to indicate mastery, 0 to indicate ignorance, and 0.5 to indicate partial mastery.

Although the overlay student model is simple and efficient, it can only represent the missing conceptions of the student. In many cases, just knowing the missing knowledge is not enough for effective tutoring. Now and then, the student makes errors because of his misconceptions. Therefore, a more informative student model

can not be a simple subset of domain knowledge; it should contain common errors (both missing conceptions and misconceptions) collected and compiled by classroom instructors and cognitive scientists.

The goal of the *bug model* research is to create a representational schema which can model both missing conceptions and misconceptions of a student. As shown in Figure 1.3 (b), in a bug student model, the student knowledge is no longer a subset of a domain knowledge base, because it includes a set of *bugs*, represented by `D3`, which are either missing conceptions or misconceptions. A tutoring system maintains a library of predefined bugs which are the most common missing conceptions or misconceptions of the students.

In LISP Tutor [54], for example, an ideal student model[2] and a bug library have been constructed. The ideal student model contains a set of production rules which represent the necessary domain knowledge for solving problems, whereas the bug catalogue contains a set of production rules (called buggy rules) which represent the common mistakes and poor strategies of novice programmers. Table 1.1 shows a production rule in the ideal student model and a related rule in the bug catalogue. The difference between these two rules lies on which function will be used to combine two lists into one. The ideal model uses `APPEND`, while the buggy rule uses `LIST` (which is commonly used by novice LISP programmers).

When a student is solving a programming problem, the LISP Tutor traces his solution step by step as the student enters his program. If he takes a correct step, the tutor will stay silent and wait for further input. If a step is diagnosed as an error, the tutor will search the bug catalogue for a buggy rule whose application can generate the same result as of the student's. Then the tutor can catch the bug in the student's solution by comparing the buggy rule with the correct rule in the ideal

[2]The ideal student model here is in fact the domain knowledge base of the system. The name is used here because an ideal student's knowledge should totally match the domain knowledge base.

Table 1.1 A Correct Rule and a Buggy Rule in LISP Tutor

Production Rule in Ideal Model	A Related Buggy Rule
IF the goal is to combine LIST1 and LIST2 into a single list THEN use the function APPEND and set subgoals to code LIST1 and LIST2	IF the goal is to combine LIST1 and LIST2 into a single list THEN use the function LIST and set subgoals to code LIST1 and LIST2

model. For instance, if a student's wrong solution can be reproduced by applying the buggy rule listed in the Table 1.1, then the tutor will think his bug is the use of the wrong function (`LIST`) to combine two lists, because the production rule in the ideal model uses another function (`APPEND`).

1.2.3 Pedagogical Knowledge

A successful teacher needs not only the knowledge about the domain subject that he is teaching and the student's understanding about the subject matters, but also the knowledge of teaching: selecting problems for the student to solve, monitoring and evaluating his performance, providing assistance upon request, and choosing remedial material. We call this type of knowledge *pedagogical knowledge*, which is the knowledge of instructional strategies. In a KBTS, the pedagogical knowledge is stored in the pedagogical knowledge base (PKB) of the system. The goal of designing a PKB is to provide various teaching strategies for a KBTS so that it can act like an experienced human instructor in a certain instructional situation [41].

In most of the existing KBTSs, the instructional strategies are basically implemented by two methods: the Socratic method and (or) the coaching method. While applying the Socratic method [17, 68, 69], the tutor teaches a subject by imposing successive questions to a student. It engages the student in a two-way conver-

sation and hopes he will learn the subject while he is answering the questions. For instance, the Socratic method used in WHY [24] questions a student in a way that will encourage him to reason about his misconceptions and therefore improve his knowledge. If a student gives water as the reason for growing rice in China, then the system will ask him: "Do you think any place with enough water can grow rice?" If the student has the knowledge that some areas (the North Pole, for example) with plenty water do not grow rice, he probably would realize that water is not a sufficient reason for growing rice.

The above question is derived from the following rule: if the student gives an explanation by stating one or more insufficient factors (water here), then formulate a general rule (any place with enough water can grow rice) asserting that the given factors are sufficient and ask the student if the rule is true. The reason for using this rule is to force the student to consider other causal factors (such as the weather in this example).

Coaching is another teaching strategy that has been successfully implemented on several systems [15, 31, 65]. The goal of the coaching is to help a student acquire skills and abilities for solving problems by engaging him in some activities such as a computer game. In a coaching situation, if the immediate aim of the student is to have fun, then skill acquisition is an indirect consequence [14, 31]. While a student is playing a game, the computer coach observes his performance, interrupts him if he takes a wrong path or a non-optimal action, and offers useful information or suggests new strategies. Coaches are best suited in those situations where skills are required for problem solving (e.g. to diagnose a fault in some electronic circuit). WEST [15] and WUMPUS [65] are examples of coaching programs.

The Socratic and the coaching strategies represent different approaches to communicate with the student. A new teaching strategy called *mixed-initiative tutoring* [22] can be formed if a tutoring system adopts both the Socratic and

the coaching strategies. In a mixed-initiative environment, the system selects the most appropriate teaching style and switches between the Socratic and coaching strategies. Therefore, a mixed-initiative tutor could ask successive questions to guide the student to understand concepts about the subject domain (in this way, it behaves as a Socratic tutor), or look over the student's shoulder and provide help while the student is solving problems posed by the system (in this way, it works as a coach).

1.3 Summary of the Chapter

As we have discussed in Section 1.1 and Section 1.2, a KBTS is fundamentally different from the traditional CAI in the following ways:

1. In a KBTS, the course materials and pedagogical strategies are organized by AI knowledge representation techniques and are stored separately in the domain knowledge bases, student models, and pedagogical knowledge bases, whereas a traditional CAI usually uses a huge, static database that incorporates all the facts to be taught and the pedagogical actions required in a teaching process.

2. In a KBTS, a model of student performance is maintained and updated dynamically. Therefore problems and remedial comments can be generated differently for each student. However, a traditional CAI has a very little information about a student performance.

3. A KBTS is able to diagnose a student's performance based on the student model and his responses to the questions given by the system and help him correct his errors, whereas a traditional CAI can only provide predefined remedial actions.

4. A KBTS attempts to solve sophisticated problems posed by a student, while a traditional CAI can only check a student's solutions with those stored in the database.

However, in the development of SQL–TUTOR [82, 83, 84], a KBTS for teaching SQL (Structured Query Language) [53, 74] to the students, we found that the current KBTS research suffers from the following drawbacks:

1. The system control mechanism is not clearly defined. Since KBTS research has been initiated primarily to explore the capability of AI techniques in the process of learning and teaching, the KBTS projects have focused on the knowledge representation issues (domain knowledge, student model, pedagogical knowledge, etc.), rather than on the control flows among the system components. Typically, most KBTSs are research prototypes that focus on only one or two components of their systems; this obscures the need for coordination among the components and the need for sophisticated control. As a result, how the various types of knowledge can be used to accomplish a tutoring task has not been fully addressed and it is still a big burden for KBTS researchers to implement an effective tutoring system.

2. Some useful knowledge about the subject domain is missing from the three system knowledge bases, namely, domain knowledge base, student model, and pedagogical knowledge base. The missing knowledge can play an important role to improve the effectiveness of a tutoring procedure. It includes:

   (a) the goal structure of a subject which describes the association of the instructional goals with the course materials. Because every course has a group of instructional goals to be accomplished, a KBTS should have the knowledge of which goal can be accomplished by teaching what topics.

   (b) the different viewpoints on the subject materials which result in different course curriculums. With this ability, a KBTS can use different curriculums for teaching different students, or for teaching the same student at different

learning stages (situations). This is a primary feature of the so called the private tutoring.

(c) the prerequisite relations among the topics of the subject. This relation determines the orders in which the topics are selected to be presented to the student, and can also help a system diagnose the mistakes made by the students.

3. Most of the existing systems lack the ability to encourage a student to play some role in determining the kind of instructional interaction which occurs. Private tutoring implies that the definition of the domain to be tutored, and the process of tutoring it, are determined by the both partners (the system and the student) of a tutoring process. If the instruction is proceeded based to the student's requests and tailored to the student's background, he is more ready to understand the domain being learned. However, for the most applications, the system insists on control from time to time and the student has vary little chance to express his own wills about the learning.

This dissertation is to investigate how these shortcomings can be overcome and to provide a framework for constructing effective KBTSs. We have focused on designing and implementing two systems. The first one is an authoring system called *Tool of Tutors (TT)* which provides instructors an environment to create KBTSs for various domains. The second one is a KBTS called SQL–TUTOR which is built by using TT for the domain of SQL programming. This dissertation is organized as follows:

Chapter 1 provides the foundation for understanding what is a KBTS, and the advantages and weaknesses of the existing KBTSs. Chapter 2 analyzes what elements constitute a tutoring process and proposes a generic architecture to implement this process. The curriculum knowledge representation is presented in Chapter 3,

which discusses what is the curriculum knowledge for a course, its importance and properties, and how it is represented in SQL–TUTOR. The discussion of curriculum knowledge management in Chapter 4 focuses on the manipulation of the curriculum knowledge stored in a KBTS so that tutoring can be tailored to a student's needs. Chapter 5 discusses the representation of the system's knowledge about the student, that is, the knowledge about the known and unknown about the student, and the use of this type of knowledge to select an appropriate topic for the student to study in the beginning of a tutoring session. Chapter 6 discusses how to represent the declarative domain knowledge in a tutoring system so that the various domain concepts and their relationships can be presented to the students effectively. Chapter 7 introduces the types of questions that can be answered by SQL–TUTOR and SQL–TUTOR's question-answering mechanism. Chapter 8 discusses the teaching of the problem solving skills, using writing SQL queries as example. Finally, Chapter 9 summarizes the contributions, limitations, as well as the future works of this research.

CHAPTER 2

A GENERIC ARCHITECTURE FOR KBTS

In Chapter 1, we overviewed the research in the field of Knowledge-Based Tutoring System (KBTS) and introduced the three major components found in a KBTS, namely, the domain knowledge base, the student model and the pedagogical knowledge base. In this chapter, we will study the characteristics of a tutoring process involving a human instructor and a student, discuss in detail what types of control procedures are necessary for a KBTS to provide effective tutoring, and introduce a generic architecture based on our discussion to support a tutoring process.

2.1 Nature of Knowledge-Based Tutoring

With a few exceptions [5, 17, 18, 45, 46, 50], most KBTS research has focused on domain knowledge representation [76, 77, 80], student model design [56, 63, 69, 72], and pedagogical knowledge representation [19, 39, 40], as opposed to the behaviors of a tutoring system during a tutoring process and the coordination of the different components inside the system to produce the desired behaviors. Because tutoring is a complicated procedure that usually includes many tasks, such as selecting a proper topic for a student, answering questions raised by the student about the topic, evaluating the student's mastery of the topic, diagnosing and helping the student correct his errors, it is necessary to study not only the types of knowledge involved in a tutoring process, but also the control modules of the system so that it can work as a human instructor.

In this section, we study what elements constitute the so called *knowledge based tutoring* and how these elements should be incorporated in a tutoring session to provide meaningful and effective tutoring. Consider the following dialogue between

a human instructor (I) and a student (S) which reveals many characteristics of the knowledge based tutoring:

Phase 1:

```
I:  What topic do you want to study today?
S:  No idea.
I:  OK. Let me select a topic for you. It is TABLE.
```

The first phase of this dialogue is a necessary preliminary step undertaken so that the instructor can determine what part of the subject materials to tutor. There are two approaches applicable here: the topics to be tutored can be selected by either the student or the instructor, and the student has the priority over the instructor. The instructor will select a topic only when the student does not want to do so. The hope is that the student will be motivated to learn, more capable of building new knowledge, and better be able to integrate the new knowledge with his existing knowledge.

Phase 2:

```
I: A TABLE contains information about objects and relationships
   among objects and has a unique name within the database.

I: The data in a table is organized in rows and columns. The
   structure of a table is determined by a table schema. Each
   table in a database usually contains a different kind of
   information.

I: Do you have any questions?

S: What are the properties of a table?

I: Each table has a unique table name within a database and its
   structure is determined by a table schema. Any other questions?

S: What is a table name?
```

I: A table name is a character string which identifies the table.
Any other questions?

S: No.

This phase of the dialogue can be further divided into two sections. In the first section, the instructor presents the content associated with the selected topic to the student. In the second section, the instructor answers the questions raised by the student about the topic just presented.

Phase 3:

I: OK. Let me ask you some questions now. Consider the following tables.
What is the birthday of John Smith?

S: January 9, 1955.

I: Good. What projects is his manager working on?

S: Hmmmm..... no idea.

EMP

John	Smith	123-456	09-JAN-55	234-567	5
Frank	Wong	234-567	08-DEC-45	456-789	5
Alicia	Zelaya	345-678	19-JUL-58	234-567	4
James	Borg	456-789	11-NOV-27	-	1

DEPT

5	Research	234-567	3366
4	Administration	234-567	1256
1	Headquarters	456-789	0234

PROJ

1	ProductX	Bellaire	5
2	ProductY	Sugarland	5
3	Reorganization	Houston	1
4	Computerization	Stafford	4

DEP_LOC

1	Houston
4	Stafford
5	Bellaire
5	Sugarland

WORK_ON

123-456	1	32
234-567	2	15
345-678	3	25
234-567	1	20

In the third phase of the dialogue, the instructor attempts to evaluate the student's mastery of the topic by posing some related questions. If the student

can not answer the questions correctly, the instructor also tries to find out what has caused the student's failure. The instructor may follow this procedure to characterize the student knowledge: "since the student has answered the first question correctly, he understands the meaning of the data in a single table; however, since he can not answer the second question, he must have some difficulties to connect the related data from multiple tables."

Phase 4:

I: OK. Answer these questions. What is John Smith's manager's ID number?

S: 234-567.

I: Right. Which table contains information about the employees and their working projects.

S: WORK_ON, I think.

I: Right. What are the meanings of the first two columns of that table?

S: An employee's ID number and the project number he is working on.

I: Have you found the ID number 345-567 there?

S: Yes.

I: What are the project numbers associated with it?

S: 1 and 2.

I: What are the project names whose numbers are 1 and 2?

S: I don't know.

I: The information about a project is stored in table PROJ. Study the definition of this table and look at the data in it.

In the fourth phase of the dialogue, the instructor helps the student find the answer. In this example, the instructor tries to help the student by asking consecutive questions that will eventually lead the answer. The instructor may ask the student to re-study some material if he finds that some prerequisite knowledge is missing from the student. If the student is able to answer the question to which he has failed before, the dialogue can go back to phase 3 and the instructor may pose more questions about the topic just discussed.

To summarize, the instructor divides a teaching dialogue into a series of phases: selecting a topic to tutor, discussing the materials related to the selected topic, getting feedbacks from the students, finding out their problems and helping them when they have difficulties. We call such a period of the dialogue a *knowledge communication cycle*, because this is a two way process: the instructor not only gives lectures to the students, but also uses the feedbacks from the students to guide his teaching. In other words, the instructor communicates with the students and both of them are the partners of the communication.

Similarly, a tutoring process in an environment involving a KBTS and a student can also be viewed as a series of communication (negotiation) cycles.[1] Each communication cycle consists of four phases: planning, discussing, evaluating, and remedying (Figure 2.1). The tasks of these phases can be specified as follows:

1. *planning*: selecting a new topic for the student to study by either the student or the system.

2. *discussing*: displaying to the student the contents of the selected topic stored in the system, and answering the student's questions about the selected topic.

[1]Note that Moyse [44], Moyse and Elson-Cook [45], and Wenger [76] also viewed a tutoring process as knowledge communication, though these authors did not divide the process into cycles or each cycle into phases.

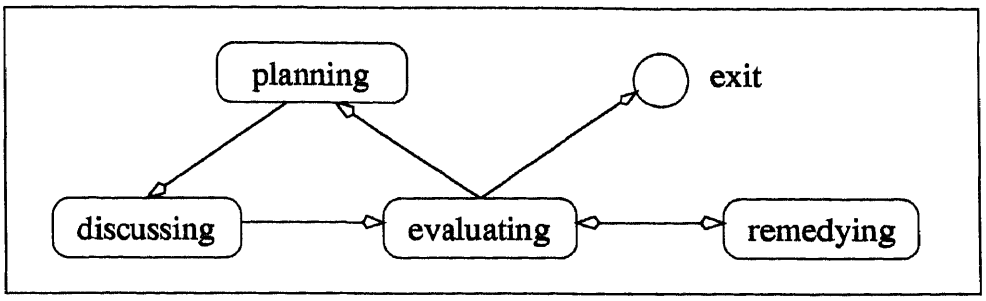


Figure 2.1 Communication Cycle in KBTS.

3. *evaluating*: posing problems associated with the selected topic to the student and evaluating his performance by analyzing his solutions.

4. *remedying*: taking any necessary pedagogical actions to correct the student's errors found in his solutions.

The system control will be passed to a remedying phase whenever a bug (a missing conception or a misconception) has been found from the student's solution during an evaluating phase. Otherwise, the system will either go back to the planning phase to start a new communication cycle, or stop the tutoring process. After the remedying, the control will be transferred back to the evaluating phase and the system will re-evaluate the student's performance. If the student still has difficulties to solve some problems posed to him, the system will continue to diagnose and remedy until his performance becomes satisfactory.

The communication cycles of a tutoring process can be nested, that is, one cycle may contain others. This kind of nesting can happen during a remedying phase if the student model shows that some prerequisite knowledge has been missing from the student. In such a case, the system may decide to suspend the current communication cycle and start a new one to re-teach a prerequisite topic to the student. Figure 2.2 portraits a tutoring session consisting of a series of communication cycles in which

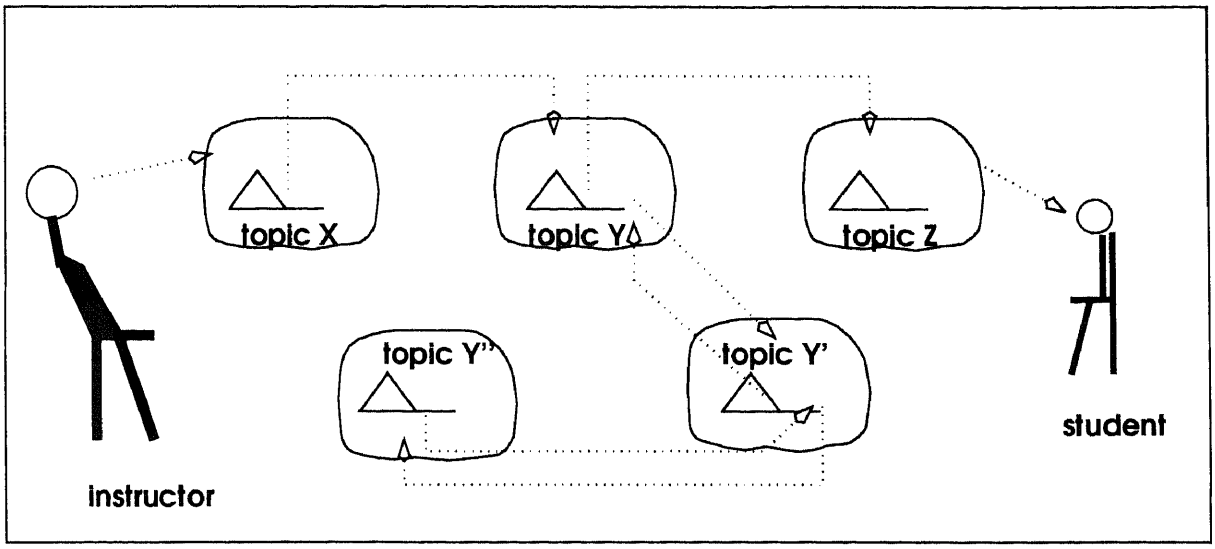


Figure 2.2 A Tutoring Process

each cycle focuses on one topic. Notice that the communication cycles for topic Y' and Y'' are nested.

2.2 System Control Mechanism

From our discussion in the previous section, we can view a tutoring process as a series of communication cycles and each of them has four major phases. Figure 2.3 shows the overall architecture of SQL–TUTOR based on our discussion.[2] Central to this architecture is the adding of the curriculum knowledge base and the communication controller. As a result, the system consists of five major components:

1. Domain Knowledge Base DKB: containing fundamental SQL concepts and their relationships that a student should learn;

2. Curriculum Knowledge Base CKB: containing the curriculum knowledge about an SQL course;

[2]For the sake of simplicity and the aid of our discussion, we use SQL–TUTOR as an example.

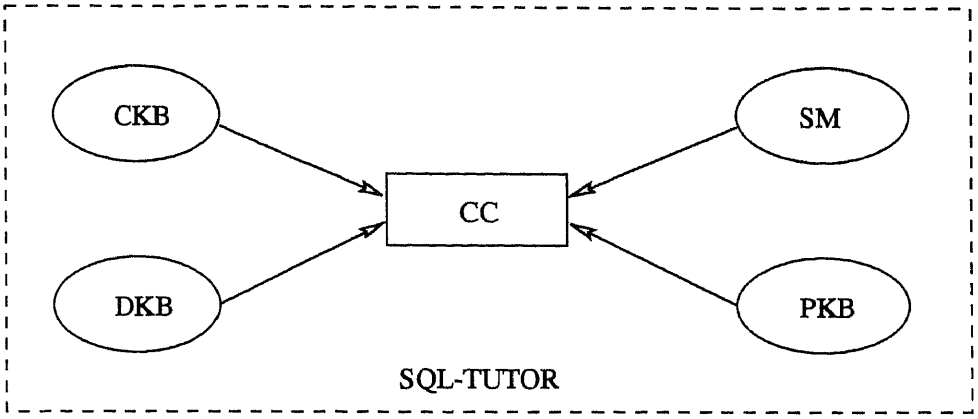


Figure 2.3 An Architecture for Knowledge-Based Tutoring System

3. Pedagogical Knowledge Base PKB: containing the knowledge of teaching SQL programming;

4. Student Model SM: containing the system's knowledge about the students' mastery level over SQL concepts and problem solving skills;

5. Communication Controller CC: containing a set of procedures controlling the system's activities during a tutoring process. These procedures are organized by four major modules:

   (a) planning module: controlling the system's behaviors during a planning phase;

   (b) discussing module: controlling the system's behaviors during a discussing phase;

   (c) evaluating module: controlling the system's behaviors during an evaluating phase; and

   (d) remedying module: controlling the system's behaviors during a remedying phase.

The functions and the design of the Communication Controller will be discussed in the remaining sections of this chapter, whereas the design of the knowledge bases (DKB, CKB, SM, and PKB) will be discussed in Chapter 3 through Chapter 8.

2.3 Planning Module

The planning module helps a student select a new topic with a proper difficulty level based on his background and performance. Figure 2.4 illustrates the components of the planning module, their input and output data, and their relationships with the system knowledge bases. There are three knowledge bases related to this module: the student model contains information about the student's mastery of the domain knowledge; the curriculum knowledge base contains the course structure and prerequisite relations among the topics; and the pedagogical knowledge base contains the knowledge of how to select a new topic from the CKB.

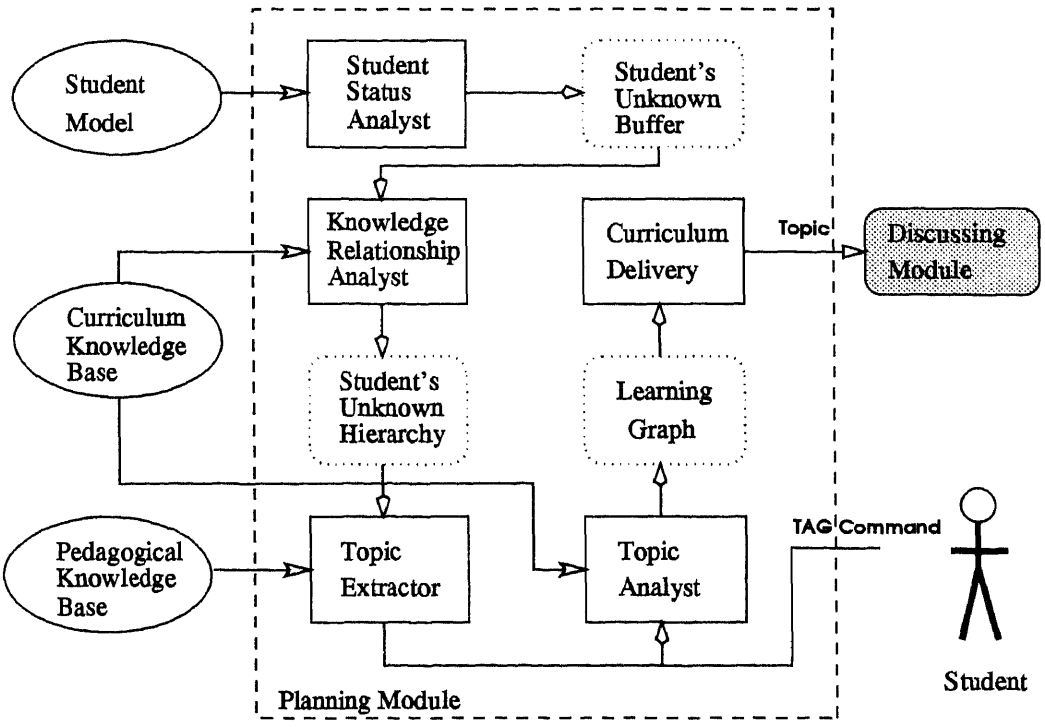


Figure 2.4 The Architecture of the Planning Module

The system can apply either an *active* or a *passive* strategy during a planning phase. In an active planning mode, the student is in the control of the selection. He can choose any topic or remove any topic from the course by issuing a TAG command. Based on the curriculum knowledge stored in the CKB, the system will suggest a set of schedules to the student for his study without forcing him to follow any of these schedules. It is up to the student to decide how to go through the course. In this mode, the student is an *actor*, who receives advice from the planning module.

During an active planning phase, after the student issues a topic to be selected or deleted, the Topic Analyst component makes reference to the CKB to generate a learning graph according to the student's requirement. Each learning graph defines a set of schedules which can guide the student's study. (A formal definition of the learning graph and its properties will be given in Chapter 4.) Then the Curriculum Delivery component traverses the generated learning graph, picks up the topics contained in the graph in certain orders, and passes them, one at a time, to the Discussing Module.

In a passive planning mode, the system is in the control of the topic selection. It executes the following procedure to select a proper topic for the student:

1. First, the Student Status Analyst has access to the student model to determine his current knowledge status (i.e. what he knows and does not know about the domain), and stores all the unknown topics into a buffer called Student Unknown Buffer;

2. Then, the Knowledge Relationship Analyst consults the curriculum knowledge base to find the prerequisite relationships among the topics in the Student Unknown Buffer and their relative difficult levels, and generates a topic hierarchy according to their relationships;

3. Next, the Topic Extractor has access to the pedagogical knowledge base to find out the strategies for determining the selection of the next topic, and passes the selected topic to the Topic Analyst;

4. Finally, the system proceeds the same procedure as we have described for the active planning mode: generating a learning graph for the selected topic and delivering the topics contained in the learning graph to the Discussing Module.

An example showing the processing of a *TAG* command entered by a student will be presented in Section 5.3, whereas an example showing the work flow of the planning module and its interactions with the knowledge bases will be presented in Section 4.5 after discussing the design of the CKB.

2.4 Discussing Module

Once a topic is received from the planning module, the discussing module is in the control. It presents the contents of the current topic to the student, and answers the questions from the student regarding this topic. Figure 2.5 depicts the structure of the Discussing Module. It proceeds according to the following procedure to accomplish its tasks:

1. First, the Domain Knowledge Extractor has access to the domain knowledge base to retrieve the contents (concepts and skills) covered by the topic and puts them in the Subject Contents Buffer;

2. Then, the Lecture Generator generates a lecture explaining the concepts or skills of the selected topic to the student;

3. On receiving the lecture, the student is allowed to ask questions about the topic, which will be analyzed by the Student Question Analyst. The internal

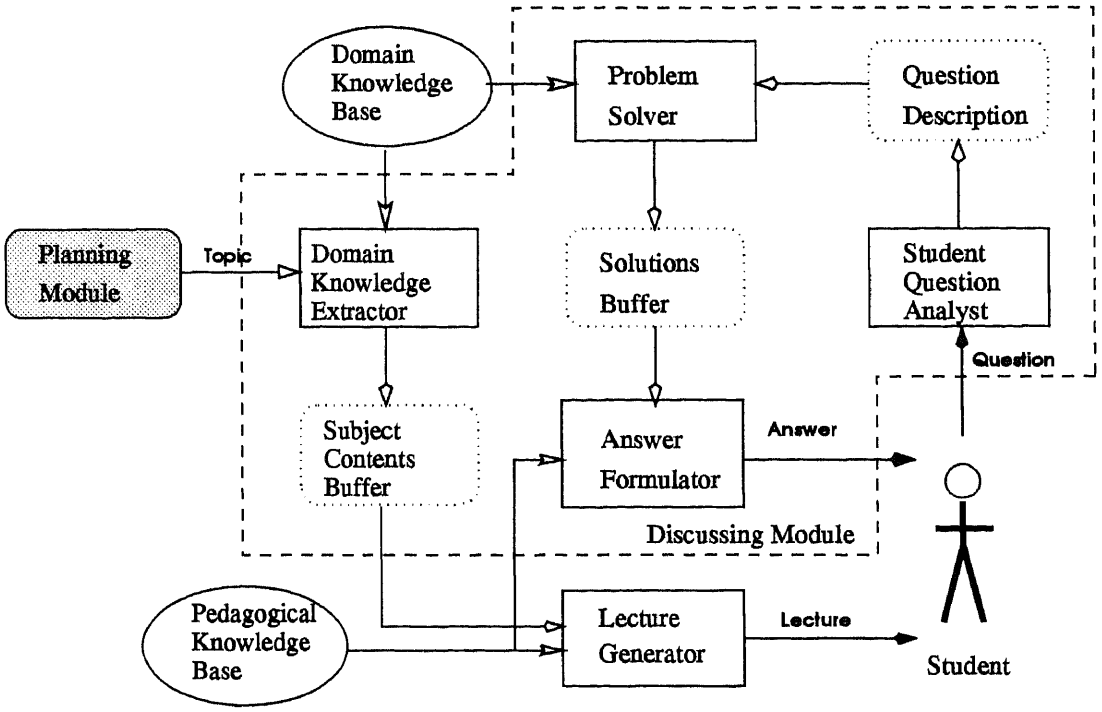


Figure 2.5 The Architecture of the Discussing Module

representation of the question is created and stored in the Question Description Buffer;

4. Next, the Problem Solver attempts to answer the questions contained in the Questions Description Buffer. For this purpose, the Problem Solver has to reason about the domain knowledge stored in the DKB. The solutions for the question will be saved in the Solution Buffer;

5. Finally, the Answer Formulator has access to the Solution Buffer to display the solutions to the student. In this way, the system has answered the questions raised by the student.

Chapter 6 discusses the representation of the declarative domain knowledge and contains an example showing the function of the discussing module. Chapter 7 describes the design of the Problem Solver and Answer Formulator. Chapter 8

discusses the representation of the procedural knowledge. Therefore, the design of the domain knowledge base DKB for a KBTS is completely covered by Chapter 6 and Chapter 8.

Since the student is allowed to ask questions using natural language in a restricted way, the Student Question Analyst must be able to understand the restricted natural language and the Problem Solver must be able to solve domain problems. Although they are very hard problems in general, we believe that they can be solved efficiently here because the question domain has been sharply limited by the topic.

2.5 Evaluating and Remedying Modules

During an evaluating phase, the evaluating module measures a student's mastery of the current topic by posing him a set of problems associated with the topic and then evaluating his responses. The structure of this module is shown in Figure 2.6. It proceeds as follows:

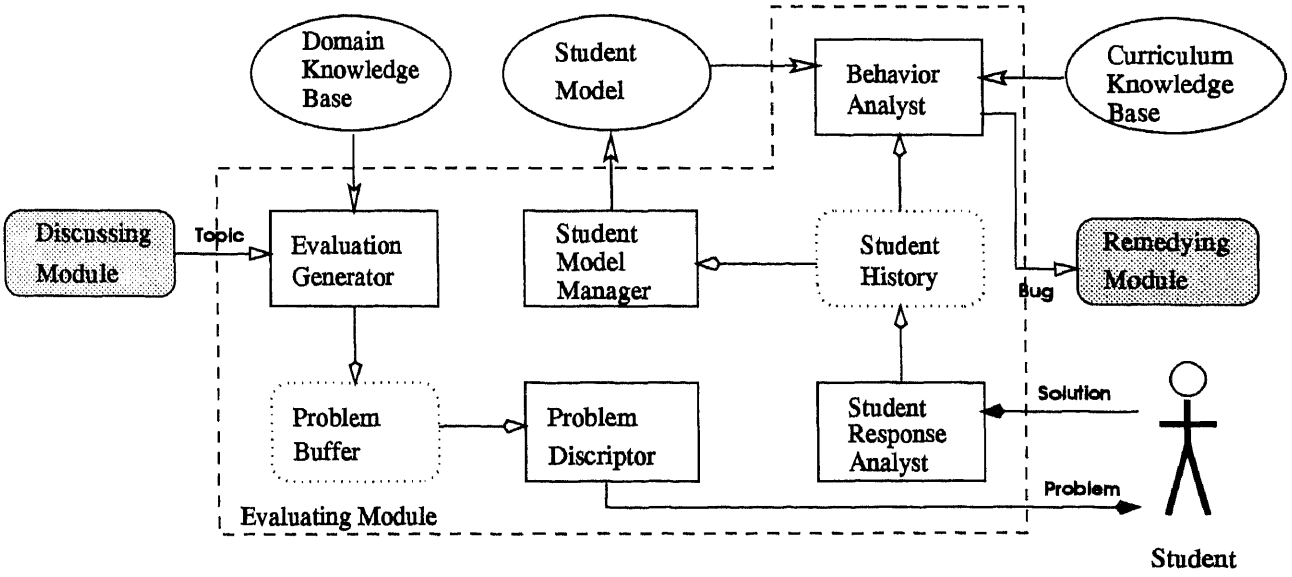


Figure 2.6 The Architecture of the Evaluating Module

1. On receiving a topic from the discussing module, the Evaluation Generator selects a set of problems associated with the topic from the domain knowledge base and stores them in the Problem Buffer;

2. Then, the Problem Descriptor converts the problems into a form which is understandable by the student, and delivers these problems to the student;

3. After that, the Student Response Analyst reads and analyzes the solution given by the student, and puts the result data from its analysis into the Student History. The result data tells whether the solution is correct, partially correct, or totally wrong;

4. Next, the Student Model Manager analyzes the data in the Student History, and updates the student model to reflect whether the student has mastered the material perfectly, or is making some progress on his learning;

5. Finally, if the student's performance is not satisfactory (e.g., he got stuck in his problem solving process, or his solution is wrong or non-optimal), the Behavior Analyst is invoked to catch the bugs in the student's knowledge which prevent him from reaching the optimal solution by analyzing the student history. In its diagnosing procedure, the Behavior Analyst may need to access the SM to find out the student's mastery of the topic and the CKB to find out the background knowledge for the current topic. Whenever a bug is found, the Behavior Analyst passes it to the Remedying Module.

During a remedying phase, the remedying module helps the student fix his bugs found in the evaluating phase. The possible remedial actions include: i) giving the student an example, a hint, a partial solution, or a complete solution; ii) letting him read the text again and concentrate on some specific parts; and iii) leading him to identify his errors by asking him some questions. The remedying strategies stored in

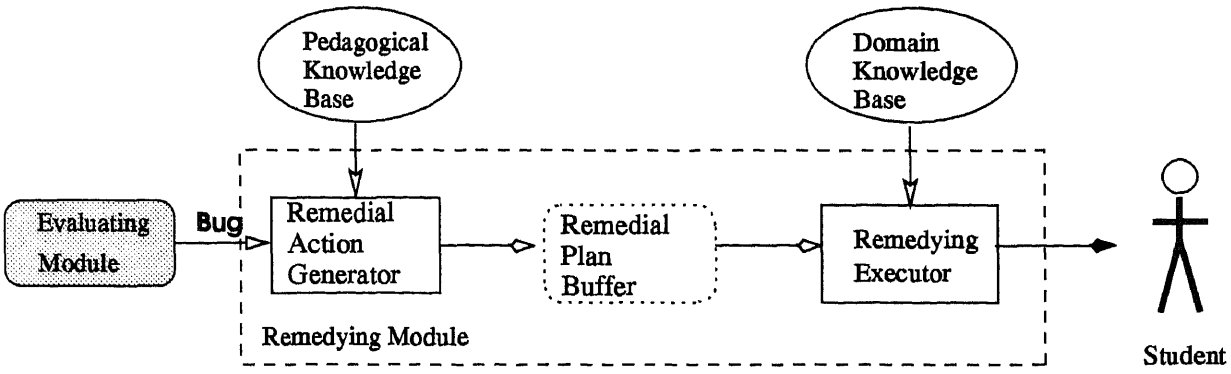


Figure 2.7 The Architecture of the Remedying Module

the PKB will determine these remedying actions. For example, one of such strategies which suggests the student to focus on a part of the domain knowledge could be:

IF the student's error is due to misunderstanding a domain concept,

THEN present and highlight the part of the domain knowledge which explains the concept.

Figure 2.7 shows the structure of the remedying module. Its proceeds as follows:

1. Based on the pedagogical knowledge stored in the PKB and the type of the bug delivered from the Evaluating Module, the Remedial Action Generator creates a remedial plan, which consists of pedagogical actions such as giving hints, showing examples, etc., and stores this executable plan in the Remedial Plan Buffer.

2. The Remedying Executor executes this plan. In the process of the execution, the executor may need the knowledge stored in the DKB. For instance, if the pedagogical rule shown in the last example has been selected by the Remedial Action Generator, the Executor will access the DKB to present some concepts.

2.6 Summary of the Chapter

Figure 2.8 shows the overall architecture envisaged from our approach. The shaded rectangles correspond to the four modules of the Communication Controller, whereas the four shaded ovals correspond to the four knowledge bases. The relationships between the knowledge bases and the modules of the Communication Controller are shown by the arcs. Our approach has the following advantages:

1. providing a generic framework for KBTS construction which can be applied to various domains;

2. incorporating the view that a tutoring process consists of a series of communication cycles, and each cycle consists of four phases and focuses on one specific topic into the design of a KBTS;

3. clarifying the knowledge communication behaviors in a tutoring process and showing what control mechanisms are needed to coordinate the system's behaviors; and

4. clarifying the roles played by the knowledge bases and their interactions with the control modules.

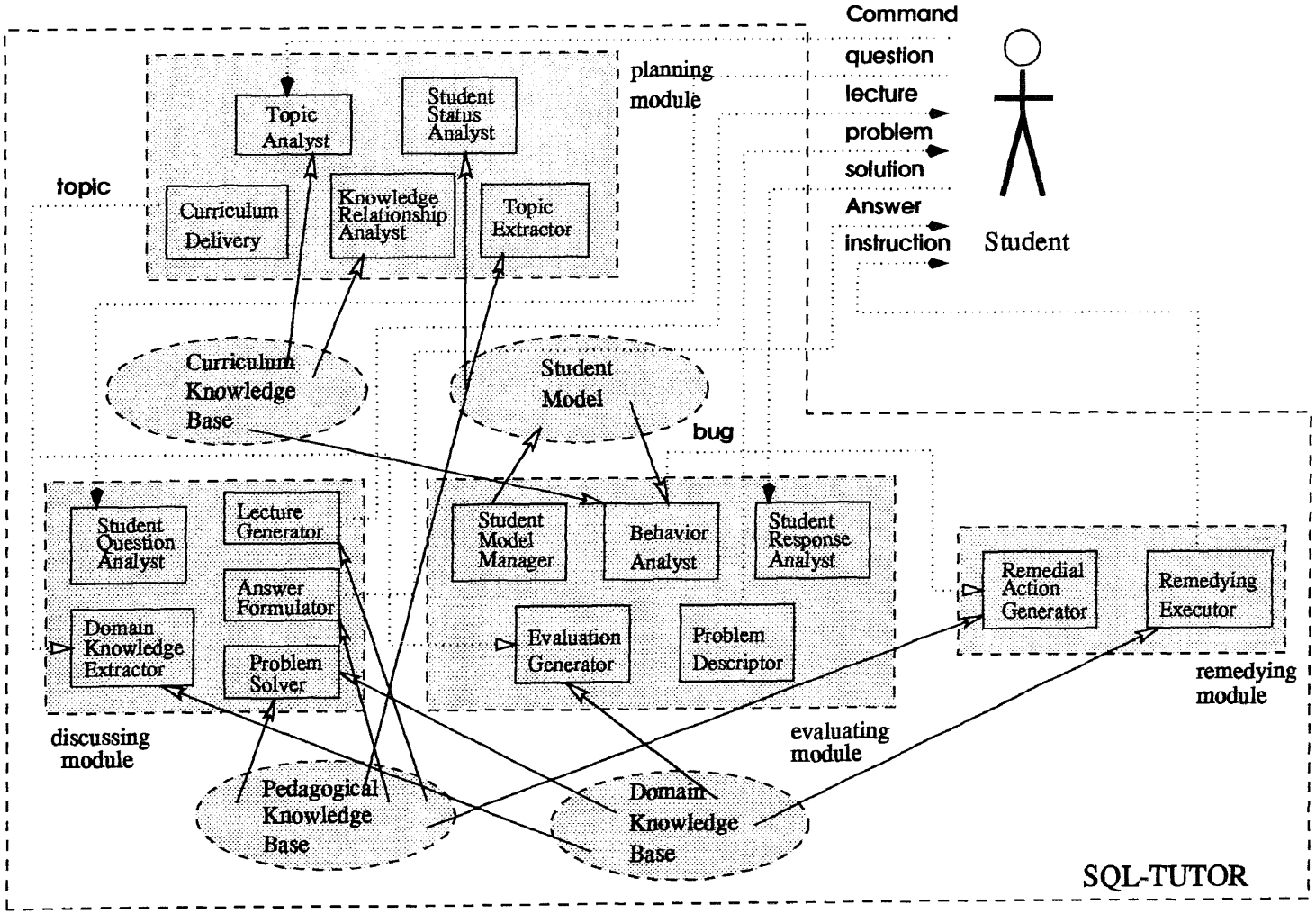


Figure 2.8 A New Architecture for Knowledge-Based Tutoring System

CHAPTER 3

CURRICULUM KNOWLEDGE REPRESENTATION

Starting from this chapter, we discuss the designs of the knowledge bases in a Knowledge-Based Tutoring System (KBTS). The major topics of this chapters are:

1. what is a teaching goal of a course and the goal hierarchy formed by the teaching goals;
2. what are the relationships among the teaching goals of a course and what are their impacts to a tutoring process;
3. how the teaching goals and their relationships can be represented in a tutoring system.

We call the knowledge about the teaching goals, the goal hierarchy, and their relationships the *curriculum knowledge*. This is the knowledge about the subject materials and is stored in the system's curriculum knowledge base CKB.

3.1 Teaching Goals and Topics

Each instructor has a set of instructional objectives to accomplish when he teaches a course. We call each of these objectives a *Teaching Goal (Tgoal)* of the course. A Tgoal describes what the students should know at the end of a tutoring session. For example, a set of possible teaching goals for tutoring SQL (Structured Query Language) can be as follows:

- have the students understand basic concepts in relational databases
- have the students learn basic SQL problem solving skills
- have the students learn querying tables

- have the students learn advanced SQL problems solving skills
- provide a foundation for further studies in database management systems.

Each of these Tgoals can be represented as a (*predicate, object*) pair, where the *predicate* is a verb describing the action to be taken and the *object* is a list of nouns or noun phrases specifying the target objects upon which the action to be applied. Using this formulation, we can rewrite the set of Tgoals described above as follows:

1. (understand, basic concepts)
2. (learn, basic SQL skills)
3. (learn, querying tables)
4. (learn, advanced SQL skills)
5. (provide, foundation)

We can further assume that:

- object "basic SQL skills" includes sub-objects "creating databases and tables", "querying tables", and "updating tables";
- object "advanced SQL skills" includes sub-objects "optimizing statements", 'creating and using views".

How can a Tgoal be accomplished through a tutoring session? It can be accomplished by attaching a list of topics to it and tutoring these topics to the students. For example, if an instructor's Tgoal is to teach the students basic concepts about relational databases, and the topic `INTRODUCTION` introduces these concepts, then the instructor can associate this topic to his Tgoal. Therefore, we can rewrite a Tgoal as (*teach, topics*), a pair of predicate and a list of topics. Table 3.1 shows the set of transformed Tgoals obtained from the original set of Tgoals given above.

Table 3.1 Two Sets of Teaching Goals

Original Tgoals	Transformed Tgoals
TO UNDERSTAND BASIC CONCEPTS IN A RELATIONAL DATABASE AND SQL	(TEACH, "INTRODUCTION")
TO INTRODUCE RELATIONAL DATA MODEL AND RELATIONAL ALGEBRA	(TEACH, "RELATIONAL DATA MODEL, RELATIONAL ALGEBRA")
TO GAIN EXPERIENCE IN USING SQL TO SOLVE PROBLEMS	(TEACH, "CREATING DATABASES AND TABLES, QUERYING TABLES, UPDATING TABLES, OPTIMIZING STATEMENTS VIEWS")
TO PROVIDE A FOUNDATION FOR FURTHER STUDIES IN DATABASE SYSTEMS	(TEACH, "AN INTRODUCTION TO SQL")

In this table, the Tgoal "provide a foundation for further studies in database management systems" is a Tgoal for the entire course; it can be achieved only by tutoring the students all the topics of the course. Therefore, we associate a special topic: `AN INTRODUCTION TO SQL`, which is the name of the course, as the object of this Tgoal.

After such a transformation, if the object of a Tgoal contains more than one topics, we can divide it into a list of equivalent Tgoals such that each Tgoal only contains one topic as its object. For example, in Table 3.1, Tgoal `(TEACH, 'OPTIMIZING STATEMENTS', 'CREATE AND USE VIEWS')` can be equivalently represented by another two Tgoals: `(TEACH, 'OPTIMIZING STATEMENTS')` and `(TEACH, 'CREATE AND USE VIEWS')`. As a consequence, each Tgoal has exactly one topic (object) and can be uniquely identified by the topic. From now on, we will use the terms topic and object interchangely if there is no confusion arises.

For each topic of a course, we associate a character string as its name and a part of the subject materials to it. We call the subject materials in a topic the *content* of the topic. Such a topic can be formally defined as follows:

Definition 1 *A topic T of a tutoring system TS is a bipartite $T = (N_T, C_T)$, where N_T is the name of the topic and C_T is a set of contents associated with the topic.* □

A topic in a tutoring system may correspond to a part, a chapter, a section, a subsection, and so forth, of a textbook, whose content consists of a subset of the subject materials. Each content in C_T is associated with a piece of text which may explain a concept in the domain (e.g., `TABLE`, `COLUMN`, `KEY`) or describe how to solve a problem (e.g., find the manager of a department). We use $C \in C_T$ to denote that C is a content of C_T. Figure 3.2 shows some topics in SQL–TUTOR and their content sets. For instance, the name of the first topic in Figure 3.2 is `CONDITIONAL RETRIEVAL`, and whose contents describe two SQL concepts: `WHERE CLAUSE` and `SEARCH CONDITION`.

Table 3.2 Some Topics in SQL–TUTOR and Their Contents.

TOPIC NAME	CONTENTS
CONDITIONAL RETRIEVAL	{WHERE CLAUSE, SEARCH CONDITION}
SIMPLE RETRIEVAL	{}
SIMPLE SEARCH EXPRESSION	{SIMPLE COMPARISON, RELATIONAL OPERATOR, PRECEDENCE RULE-1}
ONE TABLE SIMPLE RETRIEVAL	{COLUMN LIST, RESULT TABLE}
COMPOUND RETRIEVAL	{}
COMPOUND SEARCH EXPRESSION	{LOGICAL OPERATOR, PRECEDENCE RULE-2}
ONE TABLE COMPOUND RETRIEVAL	{}
MULTI-TABLE COMPOUND RETRIEVAL	{JOIN}

An important feature about the topics in a tutoring system is that they are not isolated. Instead, they are related to one another in various ways. The relationships among the topics may have a great impact on the effectiveness of the system. Therefore, it is necessary for a tutoring system to formulate the knowledge concerning the relationships among the topics. We have identified three such typical topic relations, namely, *subtopic-of*, *view-of* and *precedence-of* relations. In the following sections, we will introduce these relations and discuss in detail their representations in a tutoring system.

3.2 *Subtopic-of* Relation

A topic in a tutoring system can be usually decomposed into smaller topics, which are often called the *subtopics* of that topic, based on various factors including the Tgoals of a course to be accomplished. A subtopic, in turn, can be decomposed into even smaller topics. This process of decomposing a topic into subtopics generates a tree structure in which each node is associated with a topic, and the descendants of the node associated with its subtopics. We call such a tree a *topic tree.*

Definition 2 *A topic tree $\mathcal{T}(T)$ of a tutoring system TS is a tree rooted at topic T. Each node in a topic tree is associated with one topic from the subject domain and its descendants are associated with its subtopics.*[1] □

A leaf node $(N_T, C_T), C_T \neq \emptyset$, in a topic tree $\mathcal{T}(T)$ is called a *unit topic* (or *unit*). Topic T_1 is a subtopic of topic T_2, denoted as $S(T_1, T_2)$, if T_1 is a descendant of T_2 in $\mathcal{T}(T)$. Topic T_1 is a *child subtopic* of T_2, denoted as $CS(T_1, T_2)$, if T_1 is a child of T_2 in $\mathcal{T}(T)$. Figure 3.1 shows a partial topic tree of topic `CONDITIONAL RETRIEVAL` in SQL–TUTOR.

[1]Since each node in a topic tree is associated with exactly one topic, we can use the terminologies *node* and *topic* interchangely.

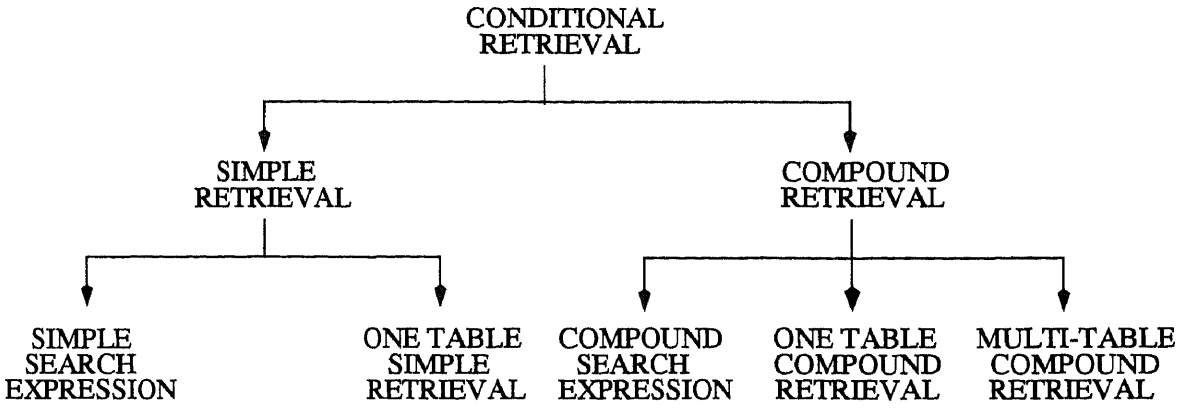


Figure 3.1 A Partial Topic Tree of CONDITIONAL RETRIEVAL.

Definition 3 *Let T be a topic of a tutoring system TS. The domain of T in TS, denoted as $Dom(T)$, is defined as follows:*

1. *if $T = (N_T, C_T)$ is a unit topic, then the domain of T is its content, that is, $Dom(T) = C_T$;*
2. *if $T = (N_T, C_T)$ is not a unit topic, then the domain of T is the union of its content and the domains of its child subtopics, that is, $Dom(T) = C_T \cup Dom(T_1) \cup \ldots \cup Dom(T_n)$, where $CS(T_i, T)$ and $1 \leq i \leq n$.*

The domain of a topic tree is the domain of its root. □

Note that if $T_1, \ldots, T_k$ are the set of subtopics of topic $T = (N_T, C_T)$, the domain of T can also be written as $Dom(T) = C_T \cup C_{T1} \ldots \cup C_{Tk}$. Table 3.3 lists the domains of CONDITIONAL RETRIEVAL and some of its subtopics (cf. Table 3.2 and Figure 3.1).

Definition 4 *A topic tree $\mathcal{T}(T)$ is well defined if for any two different topics, $T_1 = (N_{T_1}, C_{T_1})$ and $T_2 = (N_{T_2}, C_{T_2})$ of $\mathcal{T}(T)$, we have both $C_{T_1} \cap C_{T_2} = \emptyset$ and $Dom(T_1) \neq Dom(T_2)$.* □

Table 3.3 Some Topics in SQL–TUTOR and Their Domains.

TOPIC	DOMAIN
CONDITIONAL RETRIEVAL	{WHERE CLAUSE, SEARCH CONDITION, SIMPLE COMPARISON, RELATIONAL OPERATOR, PRECEDENCE RULE-1, COLUMN LIST, RESULT TABLE, LOGICAL OPERATOR, PRECEDENCE RULE-2, JOIN}
COMPOUND RETRIEVAL	{LOGICAL OPERATOR, PRECEDENCE RULE-2, JOIN}
COMPOUND SEARCH EXPRESSION	{LOGICAL OPERATOR, PRECEDENCE RULE-2}
ONE TABLE COMPOUND RETRIEVAL	{ }
MULTI-TABLE COMPOUND RETRIEVAL	{JOIN}

Therefore, in a well defined topic tree, the contents of any two distinguished topics can not be overlapped and the domains of any two distinguished topics can not be the exactly same. For a well defined topic tree, we have the following two theorems:

Theorem 1 *Let $\mathcal{T}(T)$ be a well defined topic tree and $T_1 = (N_T, C_T)$ be a node of $\mathcal{T}(T)$. If T_1 has only one child subtopic in $\mathcal{T}(T)$, then the content of T_1 is not empty, or equally, $C_{T_1} \neq \emptyset$.*

PROOF Assume that $C_{T_1} = \emptyset$ and $T_2 = (N_{T_2}, C_{T_2})$ is its only child subtopic. By Definition 3, we have $Dom(T_1) = C_{T_1} \cup Dom(T_2)$. Since we assume $C_{T_1} = \emptyset$, $Dom(T_1) = Dom(T_2)$. That is, $\mathcal{T}(T)$ is not well defined. Contradiction. □

Theorem 2 *Let $T_1 = (N_{T_1}, C_{T_1})$ and $T_2 = (N_{T_2}, C_{T_2})$ be two different topics of a well defined topic tree $\mathcal{T}(T)$. If T_1 and T_2 are not ancestors of each other in $\mathcal{T}(T)$, then their domains are disjoint, that is, $Dom(T_1) \cap Dom(T_2) = \emptyset$.*

PROOF Assume $Dom(T_1) \cap Dom(T_2) \neq \emptyset$. Let $T_{11}, \ldots, T_{1m}$ be the set of the subtopics of T_1 and $T_{21}, \ldots, T_{2n}$ be the set of the subtopics of T_2. Then $Dom(T_1) =$

$C_{T_1} \cup C_{T_{1_1}} \cup \ldots \cup C_{T_{1_m}}$ and $Dom(T_2) = C_{T_2} \cup C_{T_{2_1}} \cup \ldots \cup C_{T_{2_n}}$. If $Dom(T_1) \cap Dom(T_2) \neq \emptyset$, then there exist topics T_{1i} $(1 \leq i \leq m)$, T_{2j} $(1 \leq j \leq n)$, and a content C such that $C \in (C_{T_1} \cup C_{T_{1_i}})$ and $C \in (C_{T_2} \cup C_{T_{2_j}})$. Therefore, one of the following must be true:

1. $C \in C_{T_1}$ and $C \in C_{T_2}$;
2. $C \in C_{T_1}$ and $C \in C_{T_{2_j}}$;
3. $C \in C_{T_{1_i}}$ and $C \in C_{T_2}$;
4. $C \in C_{T_{1_i}}$ and $C \in C_{T_{2_j}}$.

Since T_1 and T_2 are not ancestors each other, $T_1 \neq T_{2j}, T_2 \neq T_{1i}$ and $T_{1i} \neq T_{2j}$. Therefore, in all the cases, there is a overlap between the contents of two different topics, that is, $\mathcal{T}(T)$ is not well defined, which is a contradiction. □

3.3 *View-of* Relation

For the given domain of a tutoring system, there are usually several ways to organize the subject materials into topic trees based on various factors. As an example, Figure 3.2 shows two topics trees rooted at topic `QUERYING TABLES`. In the first topic tree, the content of `QUERYING TABLES` is organized by the types of the `WHERE` clause of a `SELECT` statement, therefore it has subtopics `BASIC CONCEPTS`, `UNCONDITIONAL RETRIEVAL` (writing `SELECT` statements without `WHERE` clauses), and `CONDITIONAL RETRIEVAL` (writing `SELECT` statements with `WHERE` clauses). In the second organization, the content of `QUERYING TABLES` is organized by the types of the `FROM` clause in a `SELECT` statement, therefore it has subtopics `SEARCH EXPRESSION`, `ONE TABLE RETRIEVAL` (there is one table in a `FROM` clause), and `MULTI-TABLE RETRIEVAL` (there are more than one tables in a `FROM` clause).

In this section, we explore some possible relationships among these topic trees.

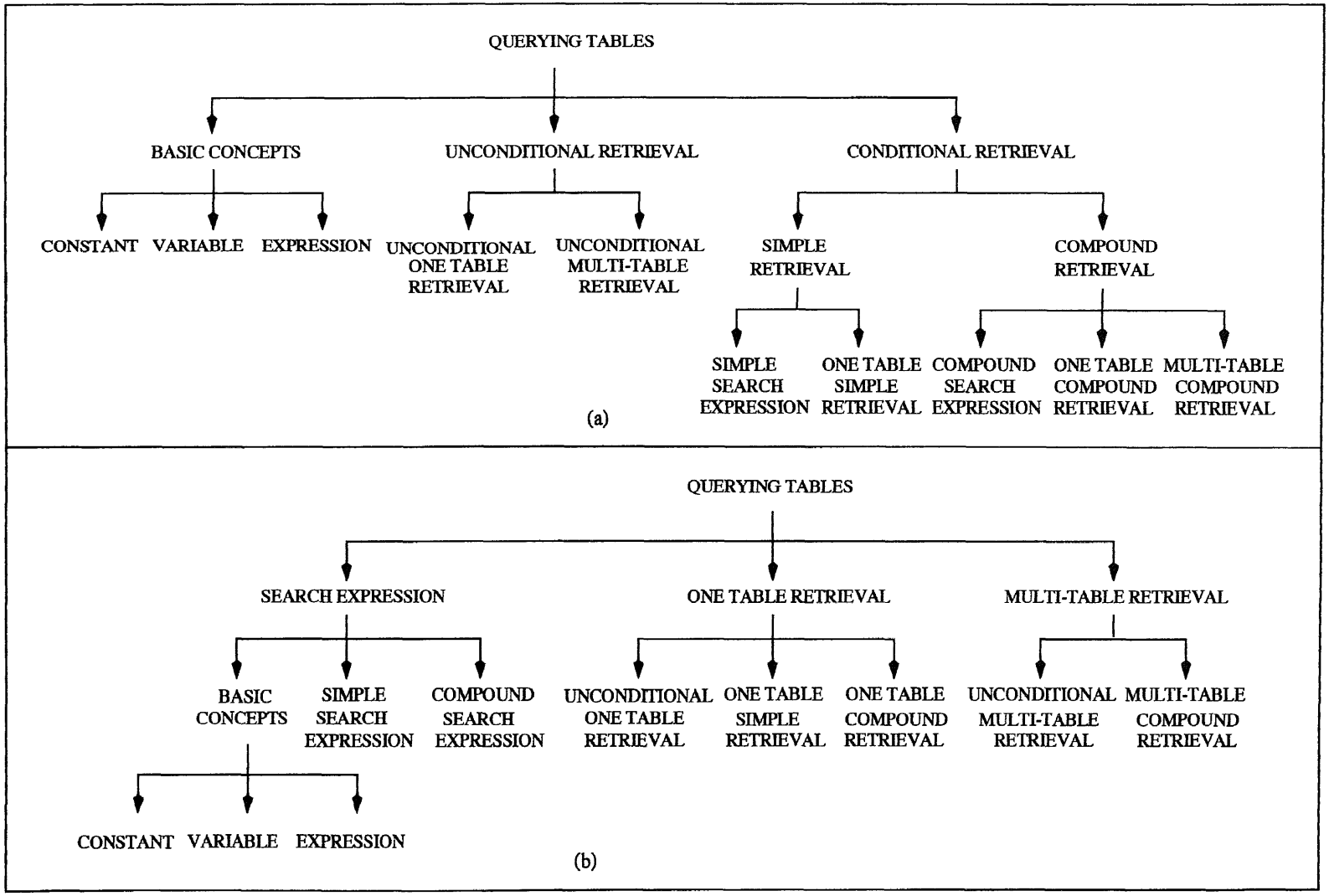


Figure 3.2 Two Topic Trees of QUERYING TABLES

Definition 5 *Let $\mathcal{T}(T_1)$ and $\mathcal{T}(T_2)$ be two well defined topic trees. $\mathcal{T}(T_1)$ and $\mathcal{T}(T_2)$ are compatible if $Dom(T_1) = Dom(T_2)$.* □

Different compatible topic trees reflect different approaches to organize the domain knowledge in a tutoring system. The capability of organizing the materials of a topic by different ways allows a system to use different approaches to achieve the set of teaching goals associated with the topic. A tutoring system with multiple topic trees in its curriculum knowledge base has two major advantages:

1. For some students, certain organization of a topic is more comfortable and effective to study than the others. It is always desirable that an instructor can select the best organization for each student. By the same token, a tutoring system which can provide multiple organizations for the subject materials can improve the teaching effectiveness, because it can adopt different organizations, based on individual needs of the students, to teach different students the same topic.

2. The system can teach a student the same topic by using different organizations in different situations. For example, a student can use one organization provided by the system to study a topic for the very first time, and then use a different organization to review the topic. In this way, the student can study the same materials from the different approaches and has a better understanding of what he is learning.

We call each way of organizing the materials of a topic a *view* of the topic. Formally, a view can be defined as follows:

Definition 6 *A view of topic T is a well defined topic tree of T.* □

We use $S(T_1, T_2, \mathcal{T}(T))$ to denote that topic T_1 is a subtopic of T_2 with respect to the view $\mathcal{T}(T)$. When the context is clear, we still use $S(T_1, T_2)$ to denote that T_1 is a subtopic of T_2 regardless of the view under which this relation holds.

Figure 3.4 shows how multiple views can be associated with nodes. In this figure, node F has two views. The first view consists of nodes K and L, whereas the second view consists of node M. Numerical labels within the nodes are used to distinguish multiple views of the nodes. Therefore, the two views of F are written as F1 = {K,L} and F2 = {M}, respectively. The following table lists the views of topics A, B and D:

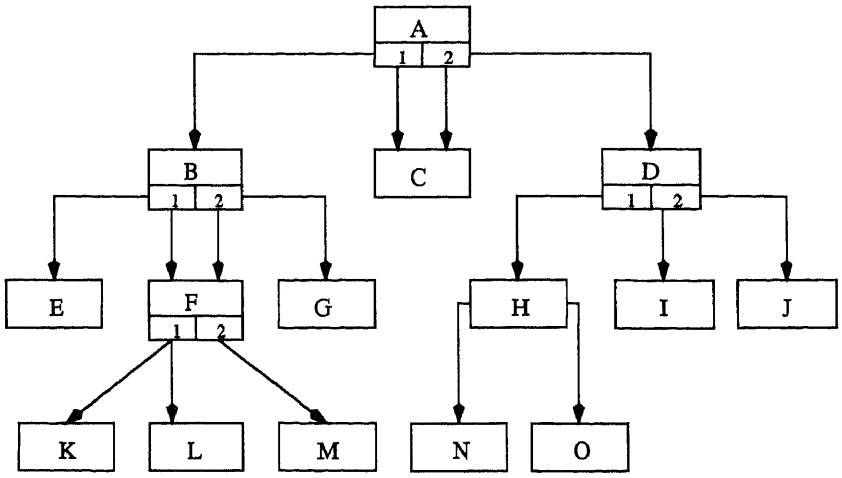


Figure 3.3 Examples of Topics and Multiple Views

A1-B1-F1 = {A,B,C,E,F,K,L}	A1-B1-F2 = {A,B,C,E,F,M}
A1-B2-F1 = {A,B,C,F,G,K,L}	A1-B2-F2 = {A,B,C,F,G,M}
A2-D1 = {A,C,D,H,N,O}	A2-D2 = {A,C,D,I,J}
B1-F1 = {B,E,F,K,L}	B1-F2 = {B,E,F,M}
B2-F1 = {B,F,G,K,L}	B2-F2 = {B,F,G,M}
D1 = {D,H,N,O}	D2 = {D,I,J}

3.4 *Precedence-of* Relation

In addition to the topic-subtopic and view relationships, there is another kind of curriculum relationship among domain concepts and topics, the *precedence-of*

relationship. Roughly speaking, content C_1 is a precedence of content C_2, denoted as $P(C_1, C_2)$, if C_1 is used to define (describe, or explain) C_2 and therefore, C_1 should be taught before C_2. In the following definition, the precedence relationships among a set of topics is formally defined based on the precedence relations among their contents.

Definition 7 *Let* $T_1 = (N_{T_1}, C_{T_1})$ *and* $T_2 = (N_{T_2}, C_{T_2})$ *be two different topics in a view* $\mathcal{T}(T)$ *of a tutoring system.* T_1 *is a precedence of* T_2 *in* $\mathcal{T}(T)$, *denoted as* $P(T_1, T_2, \mathcal{T}(T))$, *if one of the following is true:*

1. *(first order precedence) there are two contents,* C_1 *and* C_2, *such that* $C_1 \in C_{T_1}$, $C_2 \in C_{T_2}$ *and* $P(C_1, C_2)$;

2. T_1 *and* T_2 *are not ancestors of each other in* $\mathcal{T}(T)$ *and there is a topic* T' *such that* $S(T', T_1, \mathcal{T}(T))$ *and* $P(T', T_2, \mathcal{T}(T))$ *(Figure 3.4 (a));*

3. T_1 *and* T_2 *are not ancestors of each other in* $\mathcal{T}(T)$ *and there is a topic* T' *such that* $S(T', T_2, \mathcal{T}(T))$ *and* $P(T_1, T', \mathcal{T}(T))$ *(Figure 3.4 (b)).* □

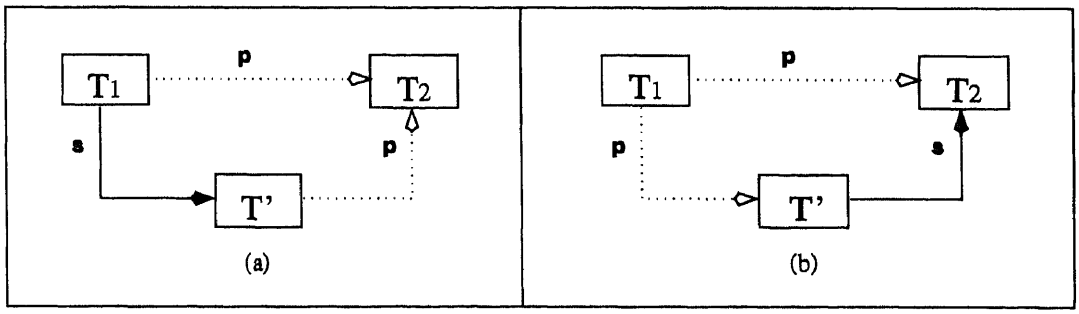


Figure 3.4 Topic T_1 Is a Precedence of Topic T_2

We use $P(T_1, T_2)$ to denote that topic T_1 is a precedence of topic T_2 when we do not concern under which view this relation holds. Consider Table 3.2 and Figure 3.1.

If both contents SIMPLE COMPARISON and RELATIONAL OPERATOR are precedences of concept LOGICAL OPERATOR, we obtain the following precedence relations:

1. P(SIMPLE EXPRESSION, COMPOUND SEARCH EXPRESSION) (first order precedence).
2. P(SIMPLE RETRIEVAL, COMPOUND SEARCH EXPRESSION) (Item 2 of Definition 7).
3. P(SIMPLE RETRIEVAL, COMPOUND RETRIEVAL) (Item 3 of Definition 7).

The precedence relationship can be used by the system to determine:

- the order in which topics and concepts to be selected and presented to the students during a tutoring process; and
- the missing prerequisite knowledge if the student can not find the correct answer of a problem.

For the precedence relations, we have the following results.

Theorem 3 *Let T_1, T_2, T_3 and T_4 be topics of the topic tree $\mathcal{T}(T)$. If $S(T_1, T_3)$, $S(T_2, T_4)$, and $P(T_1, T_2)$, then $P(T_3, T_4)$.*

PROOF Since $S(T_1, T_3)$ and $P(T_1, T_2)$, we have $P(T_3, T_2)$ (item 2 of Definition 7). Consider the fact that $S(T_2, T_4)$, we have $P(T_3, T_4)$ (item 3 of Definition 7). □

Theorem 4 *Topic T_1 is a precedence of T_2 in a topic tree $\mathcal{T}(T)$ if and only if there exist two contents, C_1 and C_2, such that $C_1 \in Dom(T_1), C_2 \in Dom(T_2)$ and $P(C_1, C_2)$.*

PROOF Let $T_1 = (N_{T_1}, C_{T_1})$ and $T_2 = (N_{T_2}, C_{T_2})$. Assume $P(T_1, T_2)$, we prove that there are two contents C_1 and C_2 such that $C_1 \in Dom(T_1), C_2 \in Dom(T_2)$ and $P(C_1, C_2)$ by induction on the sum of the heights of T_1 and T_2 in view $\mathcal{T}(T)$.

1. If HEIGHT(T_1) + HEIGHT(T_2) = 2, then neither T_1 nor T_2 has any subtopic, because both of them are unit topics. By Item 1 of Definition 7, there must be two concepts C_1 and C_2, $C_1 \in C_{T_1}, C_2 \in C_{T_2}$ and $P(T_1, T_2, \mathcal{T}(T))$. Because $C_{T_1} \subset Dom(T_1)$ and $C_{T2} \subset Dom(T_2)$, we have $C_1 \in Dom(T_1)$ and $C_2 \in Dom(T_2)$.

2. Assume the theorem is true when HEIGHT(T_1) + HEIGHT(T_2) $< k$.

3. If HEIGHT(T_1) + HEIGHT(T_2) $= k$, then $P(T_1, T_2)$ must come from one the three cases:

   (a) there are two domain concepts, C_1 and C_2, such that $C_1 \in C_{T_1}$, $C_2 \in C_{T_2}$, and $P(C_1, C_2)$. Since $C_{T_1} \subset Dom(T_1)$ and $C_{T_2} \subset Dom(T_2)$, we have $C_1 \in Dom(T_1)$ and $C_2 \in Dom(T_2)$.

   (b) T_1 and T_2 are not ancestors of each other and there is a topic $T' = (N_{T'}, C_{T'})$ such that $S(T', T_1)$ and $P(T', T_2)$. Because $S(T', T_1)$, we have HEIGHT(T') <HEIGHT(T_1). Therefore, HEIGHT(T') + HEIGHT(T_2) $< k$. From the induction assumption, there are two concepts, $C_1 \in C_{T'}$ and $C_2 \in C_{T_2}$, such that $P(C_1, C_2)$. Since $C_{T'} \subset Dom(T_1)$ and $C_{T_2} \subset Dom(T_2)$, we have $C_1 \in Dom(T_1)$ and $C_2 \in Dom(T_2)$.

   (c) T_1 and T_2 are not ancestors of each other and there is a topic $T' = (N_{T'}, C_{T'})$ such that $S(T', T_2)$ and $P(T_1, T')$. The proof is similar to (b) and omitted here.

Now suppose that there are two concepts C_1 and C_2 such that $C_1 \in Dom(T_1)$, $C_2 \in Dom(C_2)$, and $P(C_1, C_2)$. Since $C_1 \in Dom(T_1)$, there is a topic $T' = (N_{T'}, C_{T'})$ such that $S(T', T_1)$ and $C_1 \in C_{T'}$. Similarly, there is a topic $T'' = (N_{T''}, C_{T''})$ such that (T'', T_2) and $C_2 \in C_{T''}$.

1. If $T_1 = T'$ and $T_2 = T''$, then $P(T_1, T_2)$; first order precedence

2. If $T_1 = T'$ and $T_2 \neq T''$, then $P(T_1, T_2)$; Item 2 of Definition 7

3. If $T_1 \neq T'$ and $T_2 = T''$, then $P(T_1, T_2)$; Item 3 of Definition 7

4. If $T_1 \neq T'$ and $T_2 \neq T''$, then $P(T_1, T_2)$. Theorem 3

In all the cases, $P(T_1, T_2)$. □

Theorem 5 *Let $\mathcal{T}_1(T)$ and $\mathcal{T}_2(T)$ be two views. If there are two topics T_1 and T' such that $P(T_1, T', \mathcal{T}_1(T))$ and $S(T_1, T, \mathcal{T}_1(T))$, then there must be a topic T_2 such that $P(T_2, T', \mathcal{T}_2(T))$ and $S(T_2, T, \mathcal{T}_2(T))$.*

PROOF If $S(T_1, T, \mathcal{T}_2(T))$, then let $T_2 = T_1$ and the theorem is true. Otherwise, from Theorem 4, there are two concepts C_1 and C' such that $C_1 \in Dom(T_1)$, $C' \in Dom(T')$, and $P(C_1, C', \mathcal{T}_1(T))$. Since $Dom(T_1) \subset Dom(T)$, we have $C_1 \in Dom(T)$. Because $\mathcal{T}_2(T)$ is also a view of T, there is a topic $T_2 = (N_{T_2}, C_{T_2})$ such that $C_1 \in C_{T_2}$ and $S(T_2, T, \mathcal{T}_2(T))$. By Theorem 4, we have $P(T_2, T', \mathcal{T}_2(T))$. □

Theorem 5 shows us that the different views of a topic are equivalent in the sense that they provide the same prerequisite knowledge to the other topics. On the other hand, Theorem 6 shows us that these views also need the same prerequisite knowledge from the other topics.

Theorem 6 *Let $\mathcal{T}_1(T)$ and $\mathcal{T}_2(T)$ be two views. If there are two topics T' and T_1 such that $P(T', T_1, \mathcal{T}(T))$ and $S(T_1, T, \mathcal{T}_1(T))$, then there must be a topic T_2 such that $P(T', T_2, \mathcal{T}_2(T))$ and $S(T_2, T, \mathcal{T}_2(T))$.*

The proof of this theorem is similar to the proof of Theorem 5.

3.5 Topic Association Graph

If we combine the precedence relations among topics and various views about the topics, we obtain a graph called the *Topic Association Graph (TAG)*. Formally, a Topic Association Graph can be defined as follows:

Definition 8 *A Topic Association Graph in a tutoring system is a quadruple TAG = ($\mathcal{N}_s$, $\mathcal{N}_c$, $\mathcal{E}_s$, $\mathcal{E}_p$), where*

- *$\mathcal{N}_s$ is the set of nodes associated with only one view;*
- *N_c is the set of nodes associated with only more than one views;*
- *$\mathcal{E}_s$ is the set of topic-subtopic relations (edges); and*
- *$\mathcal{L}_p$ is the set of precedence relations.* □

Figure 3.5 illustrates a part of the *TAG* in SQL–TUTOR's curriculum knowledge base in which the subtopic-of and the first order precedence-of relations among topics are represented by the solid and dotted edges, respectively. In this graph, there are two views associated with QUERYING TABLES: the first view consists of nodes QUERYING TABLES, BASIC CONCEPTS, UNCONDITIONAL RETRIEVAL, CONDITIONAL RETRIEVAL, and their subtopics, whereas the second view consists of nodes QUERYING TABLES, SEARCH EXPRESSION, ONE TABLE RETRIEVAL, MULTI-TABLE RETRIEVAL, and their subtopics. In this example, QUERYING TABLES is the only node with multiple views, though in general, every node in a *TAG* can have multiple views as shown in Figure 3.4.

In the following sections, we introduce the major data structures used by our system to represent a *TAG*.

3.5.1 Node Pool

All the nodes in a *TAG* are stored in a list (array) called the *Node Pool (NP)* of the *TAG*. Each entry in a *NP* corresponds to one node in the *TAG* and is defined as a record with the following fields:

1. NAME – the name of the topic associated with the node in the *TAG*. Examples of the *NP* node names in SQL–TUTOR includes QUERYING TABLE, BASIC CONCEPTS, and CONDITIONAL RETRIEVAL;

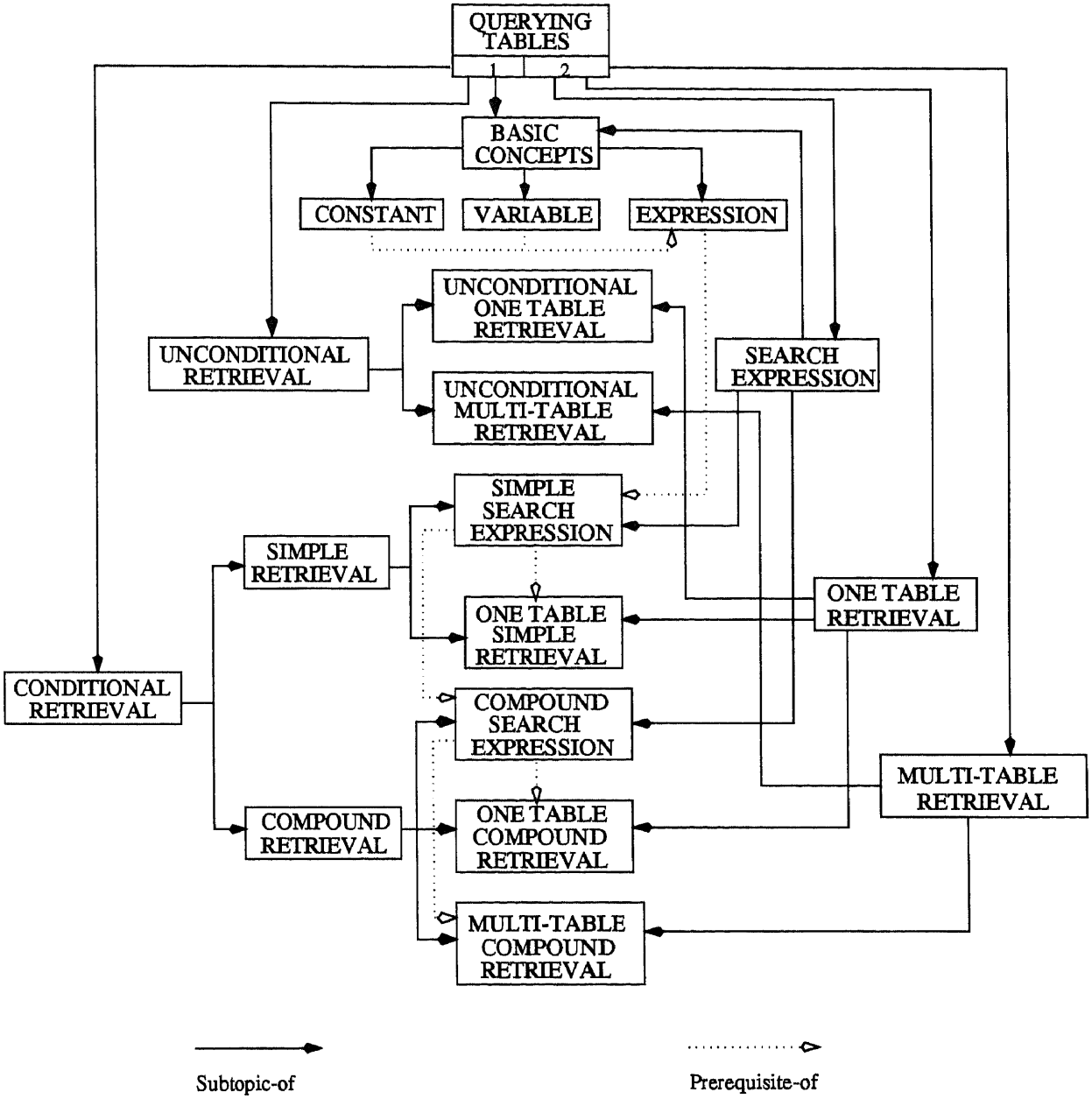


Figure 3.5 Part of Topic Association Graph (TAG) in SQL–TUTOR

2. LABEL – a unique character string used by the system to identify the node in the NP. Examples of the NP node labels in SQL–TUTOR include QT (for QUERYING TABLES), BC (for BASIC CONCEPTS), and CR (for CONDITIONAL RETRIEVAL);

3. S_1 and S_2 – the knowledge status values of the student; We will introduce them in the Chapter 5.

4. F and P – the F and P values of the node; We will introduce them in Chapter 5.

5. TOPIC – a pointer pointing to an entry (topic) in the domain knowledge base DKB which contains the subject materials associated with the topic; We will discuss the design of the topic in Chapter 6.

6. CONCEPTS – a pointer pointing a *concept list*; The concept list of a topic contains all the concepts in the content of the topic. We will discuss the design of the concept list in the Section 6.2.3.

Three functions have been defined for accessing a NP:

- `Index(label,NP)` – return the index of a node for a given NP and label.
- `Name(index,NP)` – return the name of a node for a given NP and index.
- `Label(index,NP)` – return the label of a node for a given NP and index.

3.5.2 Precedence List

The precedence relations among the contents (concepts) of a domain subject is represented by a list called the precedence list. The elements in the precedence list are pairs of concepts (C_1, C_2), which are added to the list by the instructor who creates the curriculum knowledge base CKB. (C_1, C_2) is a member of the precedence list only if C_1 is a precedence of C_2.

Since the precedence-of relations among the concepts can be represented by a DAG (Directed Acyclic Graph) in such a way that there is an edge from C_1 to C_2 if and only of (C_1, C_2) is in the precedence list, C_1 is a precedence of C_2 if and only if there is a path from C_1 to C_2 in the DAG. That is, to test the precedence relation among two concepts is a path search problem in the DAG.

In order to reduce the search complexity, we assign each concept C a unique integer as its *precedence index* (P-index) in the DAG and denote this index of C by $P_i(C)$. The indices of concepts are assigned in such a way that $P_i(C_1) \leq P_i(C_2)$ if and only if $P(C_1, C_2)$. Figure 3.6 shows a DAG which represents the precedence relations among some topics in SQL–TUTOR (cf. Figure 3.2). The integer associated with a topic is its precedence index.

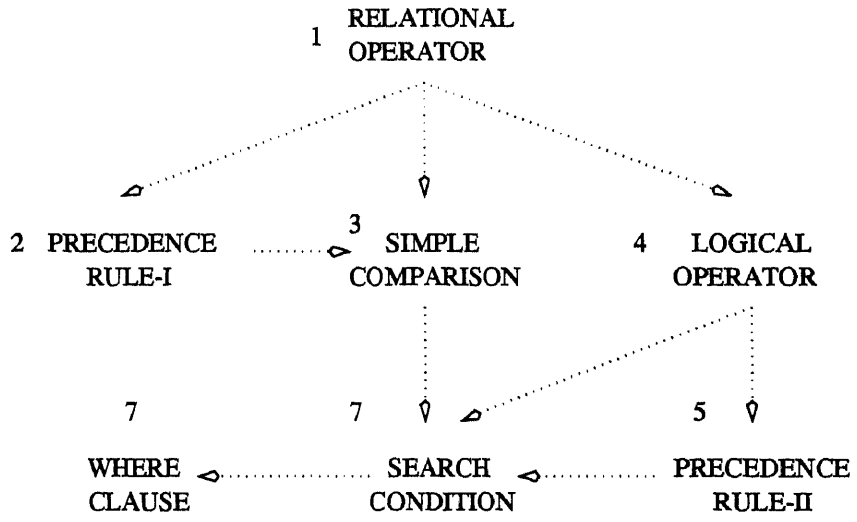


Figure 3.6 A DAG with Precedence Relations among Topics

3.5.3 Relation Matrix

The view-of and subtopic-of relations among the topics in a TAG are represented by a $N \times N$ matrix `RM`, where N is the number of topics in the TAG. Node `RM[i,j]`

represents the relationships between the TAG nodes whose indices in the node pool are i and j, respectively. Each node in a RM is a record with three fields:[2]

1. V1 – C if TAG node i is a child of node j under the first view of node j; S if TAG node i is a descendant of TAG node j under the first view of node j; empty otherwise.

2. V2 – C if TAG node i is a child of node j under the second view of node j; S if TAG node i is a descendant of TAG node j under the second view of node j; empty otherwise.

3. V3 – C if TAG node i is a child of node j under the third view of node j; S if TAG node i is a descendant of TAG node j under the third view of node j; empty otherwise.

3.5.4 TAG

A TAG is defined as a record which contains five fields (cf. Figure 3.7):

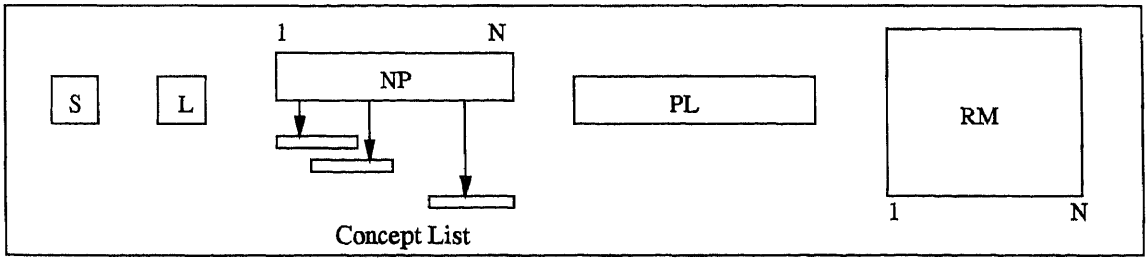


Figure 3.7 Representation of a TAG

1. S – the label of the starting node of the TAG.

2. L – the number of the nodes in the TAG.

3. NP – the NP of the TAG.

[2]We have limited the number of views which can be associated with a node to 3. That is, each node in a TAG can have at most 3 views.

4. PL – the precedence list of the TAG.

5. RM – the relation matrix of the TAG.

Figure 3.8 shows a part of the node pool, precedence list, and relation matrix for the TAG in Figure 3.5.

3.6 TAG Related Algorithms

In this section, we introduce some primary algorithms for storing information into and accessing information from a TAG:

1. IS_CHILD(i,j,TAG) – If node i is a child of node j in the TAG, return the list of views VL under which node i is a child of node j; return the empty list [ ] otherwise.

```
IS_CHILD(i,j,TAG)
{
   VL = []
   if (TAG.RM[i,j].V1 == C) then VL = VL + [1];
   if (TAG.RM[i,j].V2 == C) then VL = VL + [2];
   if (TAG.RM[i,j].V3 == C) then VL = VL + [3];
   return VL;
}
```

2. IS_SUBTOPIC(i,j,TAG) – If node i is a descendant of node j in the TAG, return the list of views VL under which node i is a descendant of node j; return the empty list [ ] otherwise.

```
IS_SUBTOPIC(i,j,TAG)
{
   VL = []
   if (TAG.RM[i,j].V1 == C || TAG.RM[i,j].V1 == S) then VL = VL + [1];
   if (TAG.RM[i,j].V2 == C || TAG.RM[i,j].V2 == S) then VL = VL + [2];
   if (TAG.RM[i,j].V3 == C || TAG.RM[i,j].V3 == S) then VL = VL + [3];
   return VL;
}
```

NODE POOL

CONDITIONAL RETRIEVAL	SIMPLE RETRIEVAL	SIMPLE SEARCH EXPRESSION	ONE TABLE SIMPLE RETRIEVAL	COMPOUND RETRIEVAL	COMPOUND SEARCH EXPRESSION	ONE TABLE COMPOUND RETRIEVAL	MULTI-TABLE COMPOUND RETRIEVAL	NAME
CR	SCR	SSE	OSR	CCR	CSE	OCR	MCR	LABEL
								CONCEPTS

PRECEDENCE LIST

RELATIONAL OPERATOR	RELATIONAL OPERATOR	RELATIONAL OPERATOR	PRECEDENCE RULE-1	LOGICAL OPERATOR	LOGICAL OPERATOR	SIMPLE COMPARISON	SEARCH CONDITION
LOGICAL OPERATOR	PRECEDENCE RULE-1	SIMPLE COMPARISON	SIMPLE COMPARISON	PRECEDENCE RULE-2	SEARCH CONDITION	SEARCH CONDITION	WHERE CLAUSE

RELATION MATRIX

	CR	SCR	SSE	OSR	CCR	CSE	OCR	MCR
CR								
SCR	{C, , }							
SSE	{S, , }	{C, , }						
OSR	{S, , }	{C, , }						
CCR	{C, , }							
CSE	{S, , }				{C, , }			
OCR	{S, , }				{C, , }			
MCR	{S, , }				{C, , }			

Figure 3.8 The Data Structure for a Topic Association Graph

3. CREATE_CHILD(i,j,V,TAG) – Make node i a child of node j under the view number V in the TAG.

```
CREATE_CHILD(i,j,TAG)
{
   switch (V) {
      case 1: {   TAG.RM[i,j].V1 == C;
                  CREATE_SUBTOPIC(i,j,1,TAG); }
      case 2: {   TAG.RM[i,j].V2 == C;
                  CREATE_SUBTOPIC(i,j,2,TAG); }
      case 3: {   TAG.RM[i,j].V3 == C;
                  CREATE_SUBTOPIC(i,j,3,TAG); }
   }
}
```

4. CREATE_SUBTOPIC(i,j,V,TAG) – Make all the descendants of node i to be the descendants of node j and all the ancestors of node j to be the ancestors of node i in the TAG.

```
CREATE_SUBTOPIC(i,j,V,TAG)
{
   for (k = 1; k <= TAG.Length; k++) {
      if (IS_SUBTOPIC(k,i,TAG) != [])
         switch (V) {
            case 1: TAG.RM[k,j,TAG].V1 = S;
            case 2: TAG.RM[k,j,TAG].V2 = S;
            case 3: TAG.RM[k,j,TAG].V3 = S;
         }
      if ((V=IS_SUBTOPIC(j,k,TAG)) != [])
         if (1 is a member of V) then
            TAG.RM[i,k,TAG].V1 = S;
         if (2 is a member of V) then
            TAG.RM[i,k,TAG].V2 = S;
         if (3 is a member of V) then
            TAG.RM[i,k,TAG].V3 = S;
         }
}
```

5. TYPE(T,TAG) – Return the type of topic T in the TAG: 0 if T is a unit topic; 1 if T is a non-unit node associated with one view; 2 if T is a non-unit node associated with multiple views.

```
TYPE(T,TAG)
{
   K1 = K2 = K3 = 0;
   j = Index(T,TAG.NP);
   for (i=1; i <= Tag.Length; i++)
      if (TAG.RM[i,j].V1 == C) {
         let K1 = 1;
         break;
      }
   for (i=1; i <= Tag.Length; i++)
      if (TAG.RM[i,j].V2 == C) {
         let K2 = 1;
         break;
      }
   for (i=1; i <= Tag.Length; i++)
      if (TAG.RM[i,j].V3 == C) {
         let K3 = 1;
         break;
      }
   K = K1 + K2 + K3;
   if (K > 1)
      return 2;
   else
      return K;
}
```

6. IS_PRECED_C(C1,C2,TAG) – If concept C1 is a precedence of concept C2 in TAG, return 1; return 0 otherwise;

```
IS_PRECED_C(C1,C2,TAG)
{
   I1 = Pi(C1);       I2 = Pi(C2);
   if (I1 > I2) then
      return 0;
   else
      for (each pairs of the form (C1,C3) in PL) {
```

```
            I3 = Pi(C3);
            if (I2 == I3) then
               return 1;
            else if (I3 > I2) then
               continue;
            else if (IS_PRECED_C(C3,C2) = 1) then
               return 1;
         }
      return 0;
   }
```

7. IS_PRECED_T(T1,T2,TAG) – If topic T1 is a precedence of topic T2 in TAG, return 1; return 0 otherwise;

```
IS_PRECED_T(T1,T2,TAG)
{
   for (each concept C1 in the CL of T1)
      for (each concept C2 in the CL of T2)
         if (IS_PRECED_C(C1,C2,TAG) == 1)
            return 1;
   return 0;
}
```

8. GET_CHILDREN(T,TAG,V) – Return the children of the topic T in the TAG under the view V.

```
GET_CHILDREN(T,TAG,V)
{
   j = Index(T,TAG.NP);     k = 1;
   for (i=1; i<=TAG.Length; i++)
      switch (V) {
         case 1: if (TAG.RM[i,j].V1 == C)
                    Cs[k++] = Label(i,TAG.NP);
         case 2: if (TAG.RM[i,j].V2 == C)
                    Cs[k++] = Label(i,TAG.NP);
         case 3: if (TAG.RM[i,j].V3 == C)
                    Cs[k++] = Label(i,TAG.NP);
      }
   return Cs;
}
```

9. IS_SP(T1,T2,TAG) – Return 1 if the topic T1 is a strong precedence of the topic T2; return 0 otherwise.

```
IS_SP(T1,T2,TAG)
{
   i1 = Index(T1,TAG.NP);
   i2 = Index(T2,TAG.NP);
   for (i=1; i<=TAG.Length; i++)
      if (is_CHILD(i,i2)) {
         T = Label(i,TAG.NP);
         if (!IS_PRECED(T,T1))
            return 0;
      }
   return 1;
}
```

CHAPTER 4

CURRICULUM KNOWLEDGE MANAGEMENT

SQL–TUTOR provides a group of commands called *TAG* operators (commands) for the students. By using these operators, a student can focus on one view of a topic and pick or skip a topic from his learning plans. In this way, the student can construct his own curriculum based on his interests and requests.

As shown in Figure 2.4, during a planning phase, the student can issue a *TAG* command to the Topic Analyst. The Topic Analyst executes the command, generates a learning graph and passes it to the Curriculum Delivery. The Curriculum Delivery traverses the learning graph and passes the topics from the graph, one at a time, to the Discussing Module. In this chapter, we discuss four major issues related to this procedure:

1. how a learning graph is defined;
2. how the *TAG* operators are defined;
3. how the *TAG* operators are implemented; and
4. how the Curriculum Delivery traverses and delivers the topics in a learning graph to the Discussing Module.

4.1 Private Tutoring

Private Tutoring is generally found to be the most effective form of instruction. In [54], Reiser and his associates reported that students working with private tutors can learn the given materials four times faster than those students who study in a classical classroom by attending lectures, reading texts, and working alone on homework problems. Bloom [4] found students working with private tutors have a

better grasp of the materials than a comparable group of students spending the same amount of time in the classroom.

The goal of KBTS research is to investigate and develop a computer tutoring system which can provide private tutoring to the students [60]. Such a system is sensitive to the following characteristics of each individual student:

1. the student's knowledge states (i.e., their known and unknowns) about the subject materials prior to and during the tutoring; and

2. the student's special learning needs, requests and interests to the subject materials.

There are two possible approaches that KBTSs can use to implement private tutoring. In the first approach, the system is at the center of the communication and takes a directive role in controlling the tutoring from the beginning to the end. Throughout the whole process of interaction, the student can raise questions, but nothing else. This is the approach taken by the most of the existing KBTSs. Such a system maintains a student model and a pedagogical knowledge base in the system. The student model contains information about the student's understanding and mastery of the domain subjects and therefore, allows a tutoring system to know the student's background and performances on the subjects and to use different teaching procedures and materials to teach different students.

The pedagogical knowledge base contains the knowledge of teaching, that is, the knowledge of what subject materials the system should present to the student, and how and when to present it. This type of knowledge allows a KBTS to apply different teaching strategies during a tutoring process. The teaching strategies can be chosen based upon several factors, including:

- the student's knowledge states maintained in the student model. For instance, a system can give more examples and explanations to a novice programmer than an experienced programmer;

- the types of topics. For example, a system can apply the coaching strategy for tutoring a student problem solving skills and apply the Socratic strategy for tutoring domain concepts; and

- the stages of problem solving. For example, a KBTS may give a very simple hint if a student has difficulty to answer a question at the first time. However, if the student has tried several times and still can not get the correct answer, then the system may offer more help.

Although this type of tutoring systems are sensitive to the students performance and can apply different teaching styles, they do not care any concerns of the students and spend little efforts to stimulate their interests of learning. Therefore, the students are very passive and hardly to be motivated to learn the subject. As a consequence, the tutoring is not very effective.

In the contrast to the first approach, the second approach of private tutoring encourages the students to engage actively in a tutoring process in such a cooperative way that the system and the student are working as a team. The system sets up the teaching objectives (e.g. "have the student know how to create a table") for tutoring topics in a course and the student selects the paths for going through the topics to achieve the teaching goals. During a tutoring process, the student have certain degree of freedom to choose or eliminate topics according to his special learning needs, requests and interests to the subject materials.

One goal of SQL–TUTOR is to implement effective private tutoring by using the second approach. As we have discussed in Chapter 2, in the active mode of a planning phase, a student is allowed to select a topic to study. After the topic has

been selected, the planning module of the Communication Controller will generate a learning graph, which is a subgraph of the original *TAG* for the course, for guiding the student to study the corresponding topic. In the following sections, we will formulate the concept of learning graph, discuss some of its properties, and show how to generate different learning graphs according to different learning objectives.

4.2 Learning Goal and Learning Graph

A learning goal (Lgoal) is a pedagogical objective of a student for a course, which can be expressed in terms of the topics of the course. Since a tutoring process is a communication process participated by both an instructor and a student in a cooperative manner, a KBTS must take the learning objectives of a student into account.

There are three kinds of Lgoals that a student can issue during a tutoring process while working with SQL–TUTOR:

1. to choose a particular topic to study (e.g., "to study how to create databases and tables");

2. to study a topic from a particular organization of the topic materials (e.g., "to study how to create databases and tables from Pratt's book [53]"); and

3. to exclude a specific topic from his curriculum (e.g., "to skip optimizing statements").

SQL–TUTOR provides its students with a group of operators (*FOCUS, STUDY, SELECT, DELETE, SKIP*) over a *TAG* to accomplish their Lgoals. By applying these operators, a student can construct his personal curriculum based on his individual requirements. Therefore, the student is relatively free to choose the best curriculum for himself.

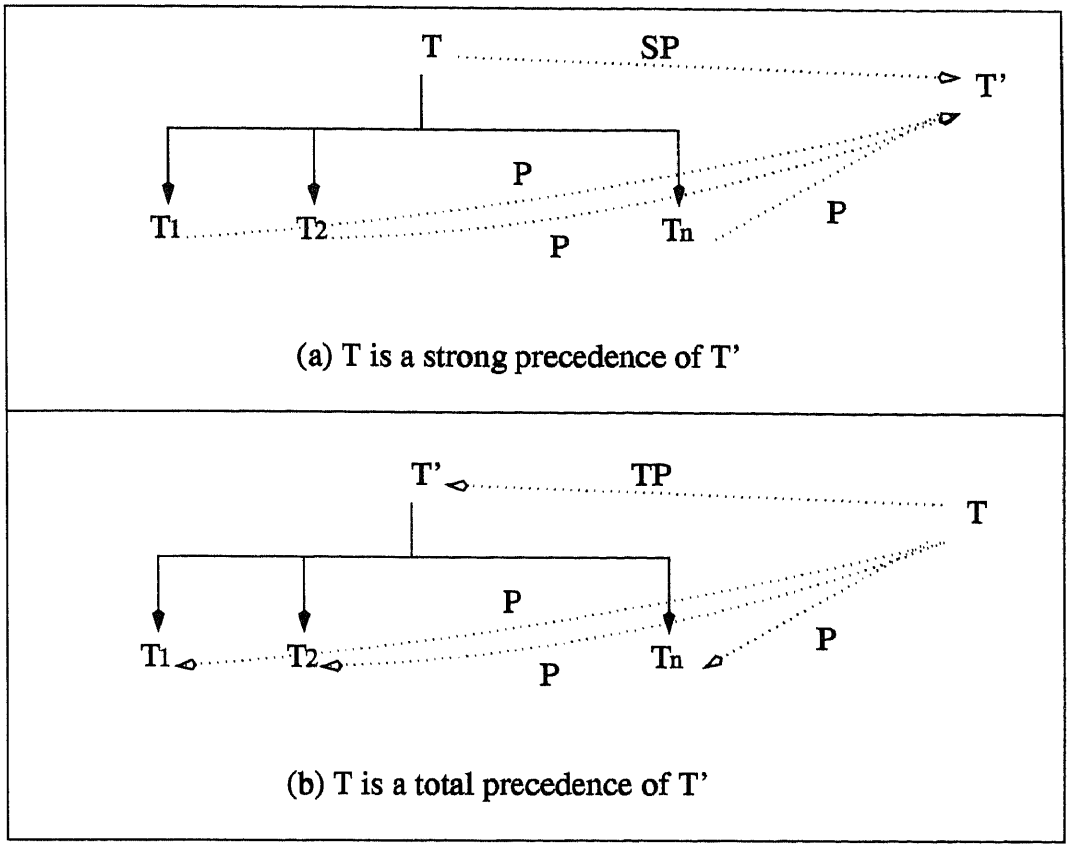


Figure 4.1 Strong and Total Precedence Relationships

Informally, each personal curriculum is called a *learning graph.* The formal definition of a learning graph is based on the concept of *strong precedence* relation.

Definition 9 *Let T and T' be two topics in a TAG and $P(T,T')$. T is said to be a strong precedence of T', denoted as $SP(T,T)$, if for all T_i, $CS(T_i,T)$ implies $P(T_i,T')$ (Figure 4.1 (a)).* □

Consider again the *TAG* shown in Figure 3.5. Topic `BASIC CONCEPTS` is a strong precedence of topic `SEARCH EXPRESSION`, because we have the following facts:

1. P(`EXPRESSION`, `SIMPLE SEARCH EXPRESSION`)

   (Given in Figure 3.5)

2. S(`SIMPLE SEARCH EXPRESSION`, `SEARCH EXPRESSION`)

(Given in Figure 3.5)

3. P(EXPRESSION, SEARCH EXPRESSION)

(1, 2, Item 3 of Definition 7)

4. P(CONSTANT, EXPRESSION)

(Given in Figure 3.5)

5. P(CONSTANT, SEARCH EXPRESSION)

(3, 4 and the transitivity)

6. P(VARIABLE, EXPRESSION)

(Given in Figure 3.5)

7. P(VARIABLE, SEARCH EXPRESSION)

(3, 6 and the transitivity)

8. SP(BASIC CONCEPTS, SEARCH EXPRESSION)

(3, 5, 7 and Definition 9)

Likewise, we can also obtain SP(BASIC CONCEPTS, SIMPLE SEARCH EXPRESSION).

Definition 10 *Let T and T' be two topics in a topic tree and $P(T,T')$. T is said to be a total precedence of T', denoted as $TP(T,T')$, if for all T_i, $CS(T_i,T')$ implies $P(T,T_i)$ (Figure 4.1 (b)).* □

Consider again the TAG shown in Figure 3.5. Topic SIMPLE RETRIEVAL is a total precedence of COMPOUND RETRIEVAL, because we have:

1. S(SIMPLE SEARCH EXPRESSION, SEARCH EXPRESSION)

(Given in Figure 3.5)

2. P(SIMPLE SEARCH EXPRESSION, COMPOUND SEARCH EXPRESSION)

(Given in Figure 3.5)

3. P(COMPOUND SEARCH EXPRESSION, ONE TABLE COMPOUND RETRIEVAL)

(Given in Figure 3.5)

4. P(SIMPLE SEARCH EXPRESSION, ONE TABLE COMPOUND RETRIEVAL)

(2, 3, and transitivity)

5. P(COMPOUND SEARCH EXPRESSION, MULTI-TABLE COMPOUND RETRIEVAL)

(Given in Figure 3.5)

6. P(SIMPLE SEARCH EXPRESSION, MULTI-TABLE COMPOUND RETRIEVAL)

(2, 5, and transitivity)

7. P(SIMPLE RETRIEVAL, COMPOUND SEARCH EXPRESSION)

(1, 2, item 3 of Definition 7)

8. P(SIMPLE RETRIEVAL, ONE TABLE COMPOUND RETRIEVAL)

(1, 4, item 3 of Definition 7)

9. P(SIMPLE RETRIEVAL, MULTI-TABLE COMPOUND EXPRESSION)

(1, 6, item 3 of Definition 7)

10. TP(SIMPLE RETRIEVAL, COMPOUND RETRIEVAL)

(7, 8, 9, item 3 of Definition 10)

Definition 11 *A learning graph $LG = (\mathcal{N}', \mathcal{E}')$ of a $TAG = (\mathcal{N}_s, \mathcal{N}_c, \mathcal{E}_s, \mathcal{E}_p)$, where $\mathcal{N}' \subset \mathcal{N}_s \cup \mathcal{N}_c$ and $E' \subset \mathcal{E}_s \cup \mathcal{E}_p$, is an induced subgraph of the TAG.* □

From this definition, a learning graph is made of a subset of the nodes and all the edges between these nodes from a TAG. Thus, given a TAG and a subset of nodes $\mathcal{N}'$ of the TAG, a learning graph $LG = (\mathcal{N}', \mathcal{E}')$ is uniquely defined.

A learning graph is *self-contained* with respect to the TAG if for any topic $T' \in \mathcal{N}'$ and $SP(T, T')$, we have $T \in \mathcal{N}'$. A TAG operator is *well defined* if it always yields self-contained learning graphs. Intuitively, a self-contained learning graph contains *all* the prerequisite contents which have to be covered before studying any topic in the graph. Thus, a student can study the topics in a self-contained learning graph without referring any contents outside.

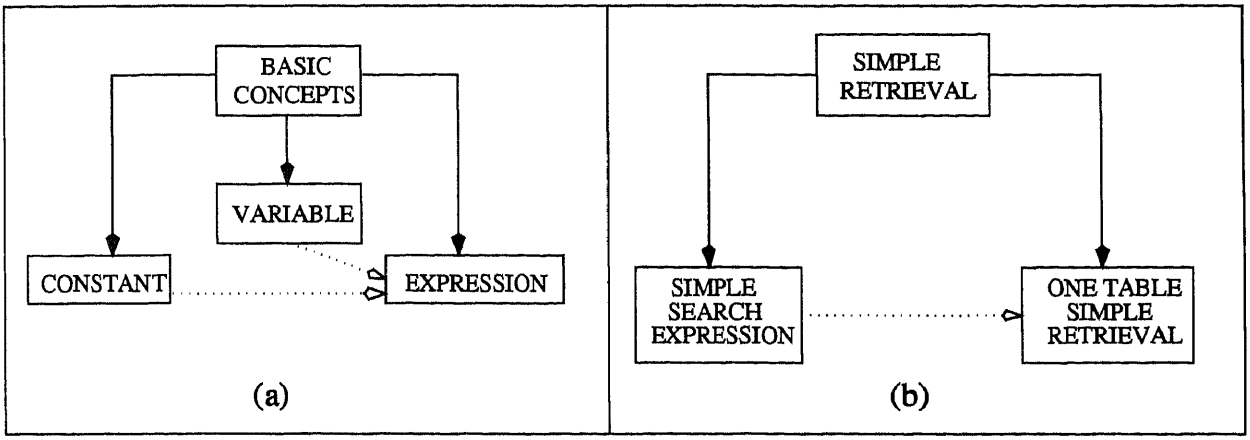


Figure 4.2 Two Learning Graphs: (a) is self-contained, but (b) is not

Figure 4.2 depicts two learning graphs generated from the *TAG* shown in Figure 3.5. The first graph is self-contained, because topic `EXPRESSION` has two prerequisite topics, `CONSTANT` and `VARIABLE`, and both of them are also members of the learning graph. Therefore, a student can learn the topics in this graph without referring to other topics. On the other hand, the second learning graph is not well defined because topic `EXPRESSION`, which is one of the precedences of `SIMPLE SEARCH EXPRESSION`, is missing from it.

4.3 *TAG* Operators

In this section, we introduce the operators which generate learning graphs from a *TAG*. We also prove that among these operators, *STUDY* and *DELETE* are well defined.

4.3.1 *FOCUS* Operator

When there are more than one views associated with a topic, a student can select a view by using *FOCUS* operator, or can ask the system to select a view for him. For the latter case, the system will make the selection based on the rules stored in its

pedagogical knowledge base PKB. For example, two of the pedagogical rules could be:

1. If $\mathcal{T}(T)$ is the view used by the student last time when he studied the topic T and he did not pass the test, then select another view this time.

2. If a view $\mathcal{T}(T)$ has been used by many students successfully, then select the view to study the topic T.

By using operator $FOCUS$, a student can select a particular view from a topic. Given a $TAG = (\mathcal{N}_s, \mathcal{N}_c, \mathcal{E}_s, \mathcal{E}_p)$, a topic $T \in \mathcal{N}_c$ which are associated with more than one views, and a view $\mathcal{T}(T)$, operation $FOCUS(TAG, \mathcal{T}(T))$ allows a student to use view $\mathcal{T}(T)$ to study topic T and discard all others. Therefore, when there are more than one view associated with a topic, the student can select his preferable one.

The result of applying operation $FOCUS(TAG, \mathcal{T}(T))$ is a learning graph $LG = (\mathcal{N}', \mathcal{E}')$, where $\mathcal{N}' = \{T' \mid S(T', T, \mathcal{T}(T))\}$. That is, $\mathcal{N}'$ is obtained from $\mathcal{N}_s \cup \mathcal{N}_c$ by removing those topics which are not subtopics of T with respect to $\mathcal{T}(T)$. Consider again the topic `QUERYING TABLES` in the TAG shown in Figure 3.5. If a student focuses on its first view labeled by `1`, then the learning graph generated is the topic tree shown in Figure 3.2 (a).

4.3.2 *SELECT* Operator

Recall that a tutoring process involving a KBTS and a student can be considered as a knowledge communication between the system and the student. This kind of communication consists of a series of communication cycles, each of which focuses on one topic. The topic can be selected by the system, or by the student using operators $SELECT$ or $STUDY$.

Given a $TAG = (\mathcal{N}_s, \mathcal{N}_c, \mathcal{E}_s, \mathcal{E}_p)$ and a topic T of the TAG, the operation $SELECT(TAG, T)$ generates a learning graph $LG = (\mathcal{N}', \mathcal{E}')$, where $\mathcal{N}' = \{T\} \cup$

$\{T' \mid S(T', T)\}$. Figure 4.3 shows the learning graph obtained by applying operator *SELECT* to topic ONE TABLE RETRIEVAL from the *TAG* shown in Figure 3.5.

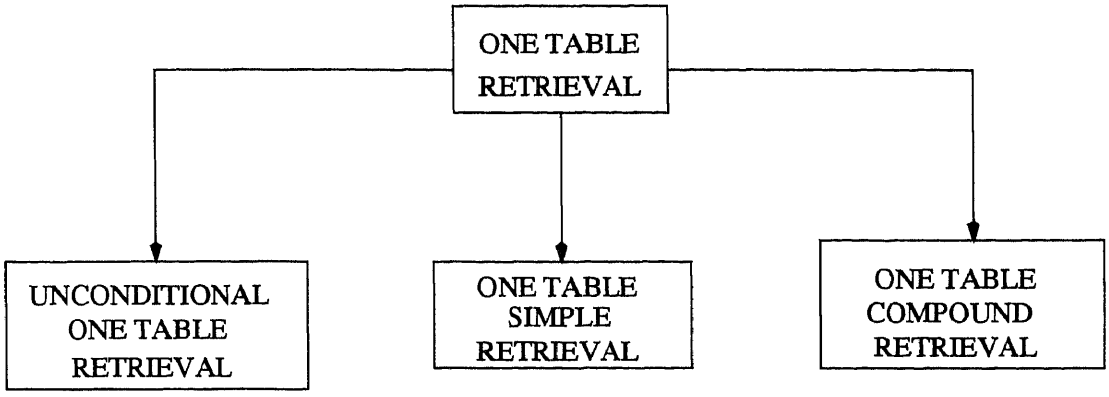


Figure 4.3 The Learning Graph Obtained by Applying Operator *SELECT*

Since the learning graph generated by a *SELECT* operation is not necessarily self-contained (i.e., *SELECT* is not well defined), the system may find that some precedence topics that the student has not mastered are not included in the learning graph. In this case, the system will tell the student that some prerequisite knowledge is missing from him and let the student decide whether he should go ahead to study the topics in the learning graph, or study the missing precedence topics first. In this way, the system works as an advisor who tells the student his knowledge status regarding the subject materials and the student can select the topic to study.

4.3.3 *SKIP* Operator

Given a $TAG = (\mathcal{N}_s, \mathcal{N}_c, \mathcal{E}_s, \mathcal{E}_p)$ and a topic T of the TAG, the operation $SKIP(TAG, T)$ yields a learning graph $LG = (\mathcal{N}', \mathcal{E}')$, where $\mathcal{N}' = (\mathcal{N}_s \cup \mathcal{N}_c) - (\{T\} \cup \mathcal{N}_1)$ and $\mathcal{N}_1 = \{ T' \mid S(T', T)\}$. That is, $\mathcal{N}'$ is obtained from $\mathcal{N}_s \cup \mathcal{N}_c$ by removing T and all its subtopics. Figure 4.4 shows the result of applying operator *SKIP* to topic SIMPLE RETRIEVAL in the *TAG* shown in Figure 3.5. Like operator *SELECT*, the learning graph generated by *SKIP* is not necessarily self-contained.

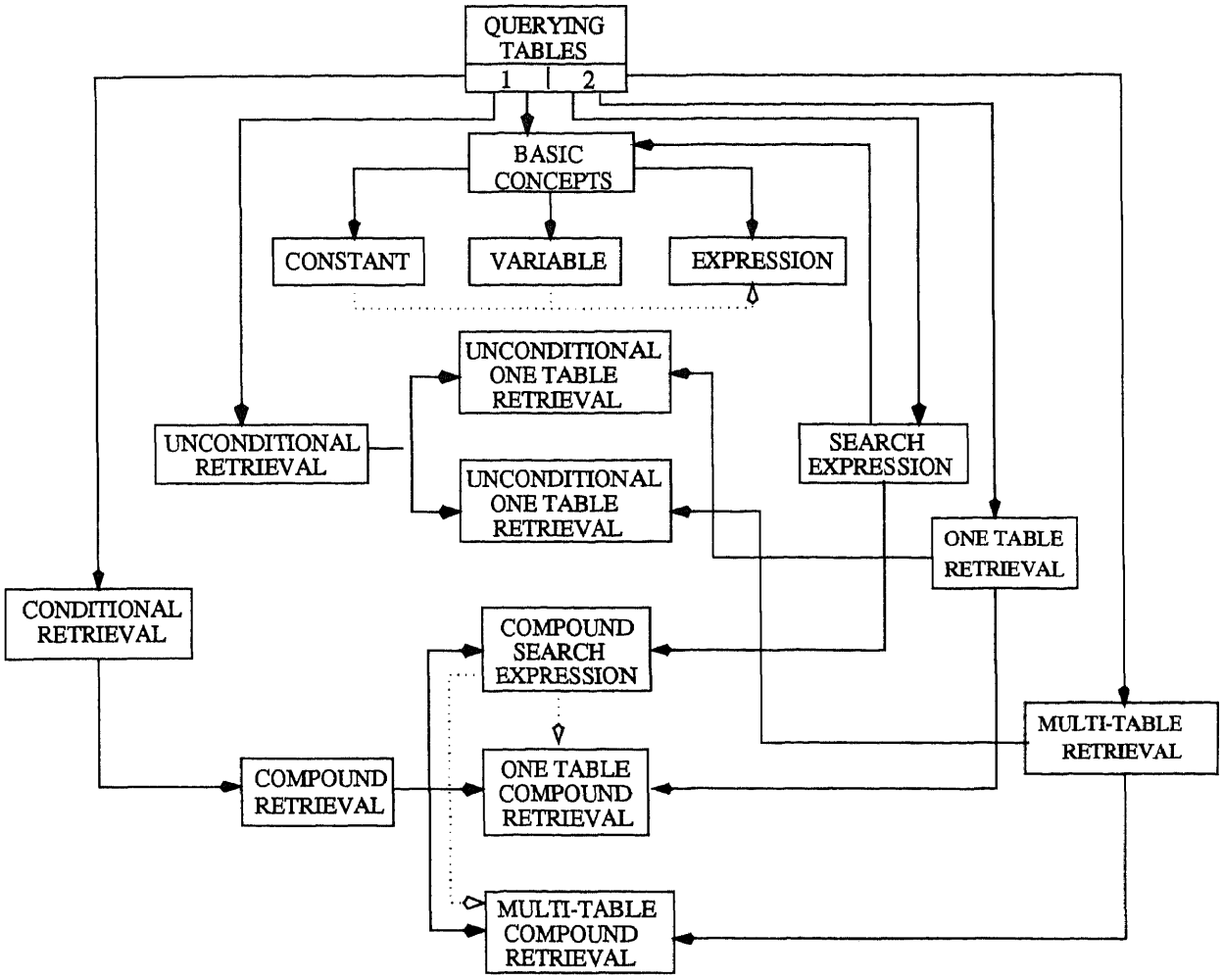


Figure 4.4 The Learning Graph Obtained by Applying Operator *SKIP*

4.3.4 *STUDY* Operator

Given a $TAG = (\mathcal{N}_s, \mathcal{N}_c, \mathcal{E}_s, \mathcal{E}_p)$ and a topic T, the operation $STUDY(TAG, T)$ creates a learning graph $LG = (\mathcal{N}', \mathcal{E}')$, where $\mathcal{N}' = \{T\} \cup \{T' \mid S(T', T) \vee SP(T', T)\}$. That is, the learning graph contains T and the topics that are subtopics of T or strong precedences of T. Therefore, the student can use this learning graph as a personal curriculum to study the selected topic.

Figure 4.5 shows the resulting graph obtained by applying *STUDY* to topic `ONE TABLE RETRIEVAL` in the TAG of Figure 3.5. In this figure, we have divided the

topics into two regions where the Region I includes ONE TABLE RETRIEVAL and all of its subtopics, whereas the Region II includes all of the topics which are strong precedences of ONE TABLE RETRIEVAL.

Theorem 7 *Operator STUDY is well defined.* □

PROOF We want to prove that for any topic $T_1 \in \mathcal{N}'$, if there is a topic T_2 such that $SP(T_2, T_1)$, then $T_2 \in \mathcal{N}'$.

1. If $T_1 = T$, then $T_2 \in \mathcal{N}'$ by the definition of $\mathcal{N}'$ in operator *STUDY*.

2. If $T_1 \neq T$, then we have either $S(T_1, T)$ or $SP(T_1, T)$. For any topic T_2, if $SP(T_2, T_1)$, by Definition 9, we have $(\forall T_3)(CS(T_3, T_2) \rightarrow P(T_3, T_1))$ (*).

   (a) If $S(T_1, T)$, then by (*) and the item 3 of Definition 7, we have $(\forall T_3)(CS(T_3, T_2) \rightarrow P(T_3, T))$. That is $SP(T_2, T)$.

   (b) If $SP(T_1, T)$, then we have $P(T_1, T)$ by Definition 9. Therefore, by (*) and the transitivity of the precedence relation, we have $(\forall T_3)(CS(T_3, T_2) \rightarrow P(T_3, T))$. That is, $SP(T_2, T)$

   In both cases, we have $SP(T_2, T)$, or $T_2 \in \mathcal{N}'$. □

Both *STUDY* and *SELECT* allow a student to select a particular topic to study, but *STUDY* differs from *SELECT* operator in that the resulting learning graph from *STUDY* includes all the precedence materials (i.e., the learning graph generated by the *STUDY* is always self-contained).

4.3.5 *DELETE* Operator

Given a $TAG = (\mathcal{N}_s, \mathcal{N}_c, \mathcal{E}_s, \mathcal{E}_p)$ and a topic T of the TAG, the operation $DELETE(TAG, T)$ allows a student to remove any optional and (or) uninteresting

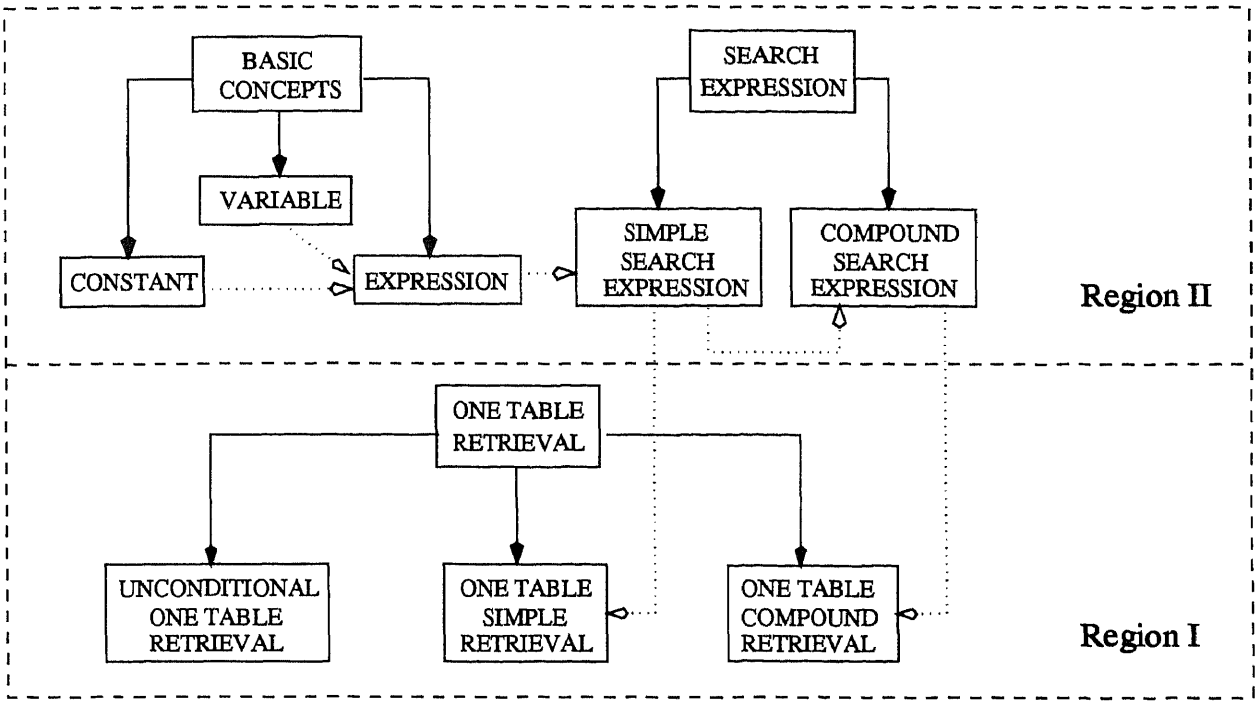


Figure 4.5 The Learning Graph Obtained by Applying Operator *STUDY*

topics from his curriculum. The operation $DELETE(TAG, T)$ generates a self-contained learning graph $LG = (\mathcal{N}', \mathcal{E}')$, where $\mathcal{N}' = (\mathcal{N}_s \cup \mathcal{N}_c) - (\{T\} \cup \mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3)$, and

1. $\mathcal{N}_1 = \{T' \mid S(T', T)\}$;
2. $\mathcal{N}_2 = \{T' \mid P(T, T')\}$; and
3. $\mathcal{N}_3 = \{T' \mid \forall T'', T''$ is a unit subtopic of T' and $T'' \in (\mathcal{N}_1 \cup \mathcal{N}_2)\}$.

That is, $DELETE(TAG, T)$ removes all the topics in $\{T\} \cup \mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3$. Intuitively, $\mathcal{N}_1$ contains all the subtopics of T; $\mathcal{N}_2$ contains topics that take T as a precedence; and $\mathcal{N}_3$ contains topics that no longer have unit subtopics due to the removal of topics in $\{T\} \cup \mathcal{N}_1 \cup \mathcal{N}_2$.

DELETE allows a student to remove some topic from his curriculum, and the learning graph generated is always self-contained. If a student wants to remove

an optional topic and everything related to the topic from his study curriculum, he can use *DELETE*. On the other hand, if the student already knows the material of a topic and wants to remove it while keeping the related materials in the study curriculum, he can use *SKIP*.

Figure 4.6 shows the result of applying *DELETE* to topic SIMPLE RETRIEVAL in the *TAG* shown in Figure 3.5. Here, we have removed the following topics:

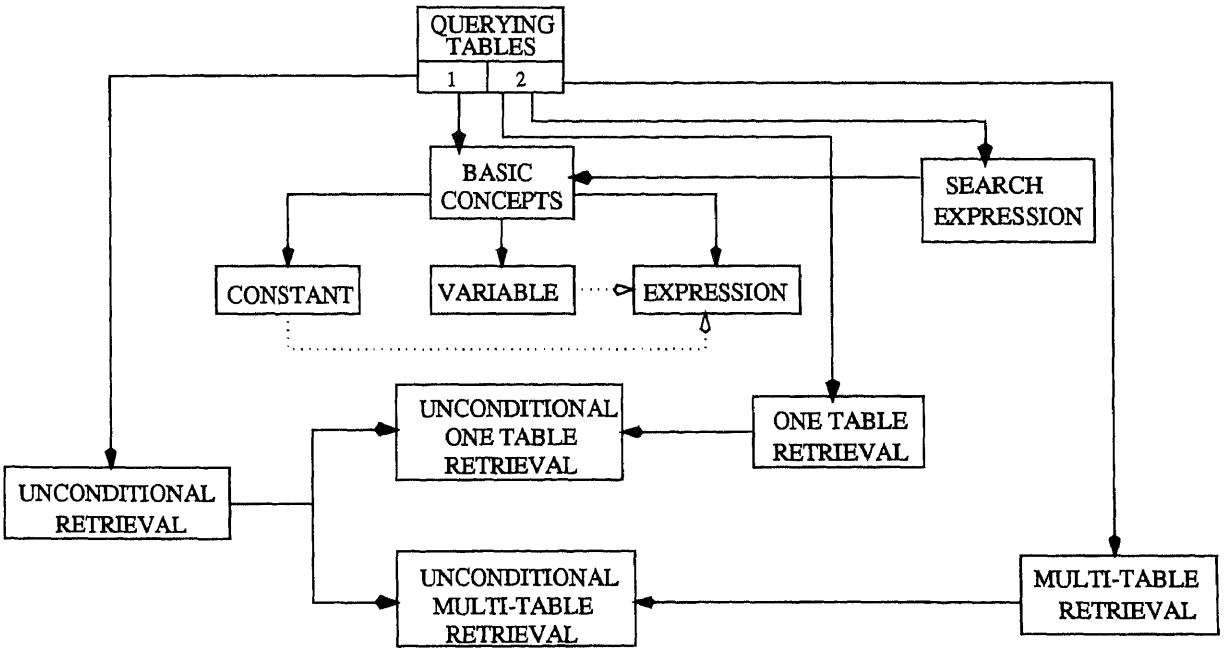


Figure 4.6 The Learning Graph Obtained by Applying *DELETE*

1. SIMPLE RETRIEVAL;
2. all the subtopics of SIMPLE RETRIEVAL;
3. COMPOUND RETRIEVAL and its subtopics, because they all take SIMPLE RETRIEVAL as their precedence;
4. CONDITIONAL RETRIEVAL, because its unit subtopics are all removed.

Theorem 8 *Operator DELETE is well defined.* □

PROOF We need to prove that for any topic $T_1 \in N'$, if there is a topic T_2 such that $SP(T_2, T_1)$, then $T_2 \in \mathcal{N}'$. Notice that by Definition 9, if we have $SP(T_2, T_1)$, then $P(T_2, T_1)$ (*).

If $T_1 \in \mathcal{N}'$, then $T_1 \notin \mathcal{N}_2$. Suppose that there is a topic T_2, $SP(T_2, T_1)$, but $T_2 \notin \mathcal{N}'$, or $T_2 \in (\{T\} \cup \mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3)$.

1. If $T_2 = T$, then $P(T, T_1)$ by (*). Therefore, $T_1 \in \mathcal{N}_2$, which is a contradiction.

2. If $T_2 \in \mathcal{N}_1$, then $S(T_2, T)$. By the item 2 of Definition 7 and (*), $P(T, T_1)$. Therefore, $T_1 \in \mathcal{N}_2$, which is a contradiction.

3. If $T_2 \in \mathcal{N}_2$, then $P(T, T_2)$. By the transitivity of the precedence relation and (*), $P(T, T_2)$. Therefore, $T_1 \in \mathcal{N}_2$, which is a contradiction.

4. If $T_2 \in \mathcal{N}_3$, then $(\forall T_3)(Unit(T_3) \wedge S(T_3, T_2) \rightarrow S(T_3, T) \vee P(T, T_3))$.

   (a) If $(\exists T_3)(Unit(T_3) \wedge S(T_3, T_2) \wedge P(T, T_3))$, then by the item 3 of Definition 7, we have $P(T, T_2)$. By the transitivity of the precedence relation and (*), $P(T, T_1)$, or $T_1 \in \mathcal{N}_2$, which is a contradiction.

   (b) If $(\forall T_3)(Unit(T_3) \wedge S(T_3, T_2) \wedge S(T, T_3))$, then $S(T_2, T)$. By the item 2 of Definition 7, $P(T, T_1)$, or $T_1 \in \mathcal{N}_2$, which is a contradiction. $\square$.

4.4 Implement TAG Operators

In this section, we introduce algorithms which implement the TAG operators. Since a learning graph LG is an induced subgraph of a TAG, it can be determined by a set of topics in the TAG. In our system, we represent a LG by a list containing the labels of the topics in LG. With this representation, given a TAG, to construct a learning graph is to create its topic list.

4.4.1 Implement *FOCUS* Operator

The algorithm FOCUS implements the *FOCUS* operator. The inputs of FOCUS are a *TAG*, a topic T of the *TAG* and a view V associated with T. It creates a list which contains the labels of topic T and all the subtopics of T under the view.

```
FOCUS(TAG,T,V)
{
   k = Index(T,TAG.NP);
   LG[1] = Label(k,TAG.NP);
   l = 2;
   for (i=1; i<=TAG.Length; i++)
      switch(V) {
         case 1: if (TAG.RM[i,k].V1 == C || TAG.RM[i,k].V1 == S)
                    LG[l++] = label(i,TAG.NP);
         case 2: if (TAG.RM[i,k].V2 == C || TAG.RM[i,k].V2 == S)
                    LG[l++] = label(i,TAG.NP);
         case 1: if (TAG.RM[i,k].V3 == C || TAG.RM[i,k].V3 == S)
                    LG[l++] = label(i,TAG.NP);
      }
}
```

4.4.2 Implement *SELECT* Operator

The algorithm SELECT implements the *SELECT* operator. The inputs of SELECT are a *TAG* and a topic T of the *TAG*. It creates a list which contains the labels of topic T and all the subtopics of T.

```
SELECT(TAG,T)
{
   k = Index(T,TAG.NP);
   LG[1] = label(k,TAG.NP);
   l = 2;
   for (i=1; i<=TAG.Length; i++)
      if (TAG.RM[i,k].V1 == C || TAG.RM[i,k].V1 == S)
         LG[l++] = label(i,TAG.NP);
      else if (TAG.RM[i,k].V2 == C || TAG.RM[i,k].V2 == S)
         LG[l++] = label(i,TAG.NP);
      else if (TAG.RM[i,k].V3 == C || TAG.RM[i,k].V3 == S)
         LG[l++] = label(i,TAG.NP);
}
```

4.4.3 Implement *SKIP* Operator

The algorithm SKIP implements the *SKIP* operator. The inputs of SKIP are a *TAG* and a topic T of the *TAG*. It creates a list which contains the labels of all the topics except T and its subtopics.

```
SKIP(TAG,T)
{
   l = 1;     k = Index(T,TAG.NP);
   for (i=1; i<=TAG.Length; i++)
      if (i != k && !IS_SUBTOPIC(i,k,TAG))
         LG[l++] = label(i,TAG.NP);
}
```

4.4.4 Implement *STUDY* Operator

The algorithm STUDY implements the *STUDY* operator. The inputs of STUDY are a *TAG* and a topic T of the *TAG*. It creates a list which contains the labels of T, the subtopics of T, and all the topics that are strong precedence of T.

```
STUDY(TAG,T)
{
   LG[1] = T;      l = 2;
   k = Index(T,TAG.NP);
   for (i=1; i<=TAG.Length; i++) {
      T1 = label(i,TAG.NP);
      if (IS_SUBTOPIC(i,k,TAG) || IS_SP(T1,T))
         LG[l++] = T1;
   }
}
```

4.4.5 Implement *DELETE* Operator

The algorithm DELETE implements the *DELETE* operator. The inputs of DELETE are a *TAG* and a topic T of the *TAG*. It creates a list which contains the labels of all the topics in *TAG* except T, the subtopics of T, and the topics of which T is a precedence.

```
DELETE(TAG,T)
{
   k = Index(T,TAG.NP);
   l = 1;
   for (each unit topic T1) {
      i1 = Index(T1,TAG.NP);
         if (i1 != k && !IS_SUBTOPIC(i1,k,TAG) && !IS_PRECED(k,i1,TAG))
            LG[l++] = T1;
   for (i=1; i<=TAG.Length; i++)
      if (i != k && !IS_SUBTOPIC(i,k,TAG) && !IS_PRECED(k,i,TAG))
         for (j=1; j<=Length(LG); j++)
            if (IS_SUBTOPIC(j,i,TAG))
               LG[l++] = label(j,TAG.NP);
}
```

4.5 Traversing a Learning Graph

After the Topic Analyst generates a learning graph, the learning graph is passed to the Curriculum Delivery. The task of the Curriculum Delivery is to select topics from the learning graph and pass them to the next phase, the discussing phase. Since a learning graph usually contains several topics and the orders in which these topics should be taught are constrained by the subtopic-of, view-of, and precedence-of relations among the topics, proper strategies are needed to select all the topics in the learning graph. In this section, we introduce the SQL–TUTOR traversal procedures and the rules to select the topics from a learning graph.

Definition 12 *Let LG be a learning graph. LG is said to have a schedule if for any two sibling T_1 and T_2 in a view $\mathcal{T}(T)$, we do not have both $P(T_1, T_2)$ and $P(T_2, T_1)$.* □

If a learning graph has a schedule, then the precedence relations over the siblings of a topic form a partial order. Therefore, the Curriculum Delivery is capable to select topics from the siblings in such a way that for any two siblings T_1 and T_2,

if $P(T_1, T_2)$, then T_1 is selected before T_2. There are three major algorithms in Curriculum Delivery:

1. The first algorithm, called T-DELIVER, takes two parameters, a TAG and a learning graph LG. It sorts the topics in the LG based on the precedence-of relation, and delivers the topics in the sorted order to the Discussing Module. This algorithm invokes the other two algorithms during its execution.

```
T-DELIVER(TAG,LG)
{
   while (LG is not empty) {
      Ts = [];
      for (each topic T in LG) {
         i = Index(T,TAG);     P = 0;
         for (j=1, j<=TAG.Length; ++j)
            if (IS_SUBTOPIC(i,j,TAG) {
               P = 1;
               break;
            }
         if (!P)
            Ts = Ts + [i];
            remove T from LG;
      }
      T-SORT(Ts,Tn,TAG);
      for (each topic T in Tn)
         TUTOR-SUBTOPICS(T,TAG);
   }
}
```

2. The second algorithm, called T-SORT, sorts a collection of related topics. Intuitively, this algorithm works by performing topological sorting on a list of topics based on the precedence-of relationships.

```
T-SORT(Ts,Tn,TAG)
{
   SORTED := [];
   while (Ts is not empty) {
```

```
      T = Ts[1];    i = Index(T,TAG.NP);
      PRECED = 0;
      for (j=1; j<=TAG.Length; ++j)
         if (IS_PRECED(j,i,TAG)) {
            PRECED = 1;
            break;
         }
      if (!PRECED) {
         remove T from Ts;
         SORTED = SORTED + [T];
      }
   }
}
```

3. Given a topic T and a TAG, the algorithm TUTOR-SUBTOPICS determines the order in which the subtopics of T are discussed and calls T-DISCUSS, which is a function in the Discussing Module, to discuss the topic with the student.

```
TUTOR-SUBTOPICS(T,TAG)
{
   K = TYPE(T,TAG);
   switch (K) {
      case 0: T-DISCUSS(T,TAG);
      case 1: { Ts = GET_CHILDREN(T,TAG,1);
                T-SORT(Ts,Tn);
                for (each topic T in Tn)
                   TUTOR-SUBTOPICS(T,TAG);
              }
      case 2: { Select a view V from T;
                Ts = GET_CHILDREN(T,TAG,V);
                T-SORT(Ts,Tn);
                for (each topic T in Tn)
                   TUTOR-SUBTOPICS(T,TAG);
              }
   }
}
```

As an example to show how the Curriculum Delivery works, consider the learning graph shown in Figure 4.5. If that learning graph is passed to T-DELIVER, then the system will follow the following steps:

1. T-DELIVER passes the un-sorted set of topics, {BASIC CONCEPTS, SEARCH EXPRESSION, ONE TABLE RETRIEVAL}, to T-SORT;

2. T-SORT sorts the passed the topics and return the sorted set of topics, {BASIC CONCEPTS, SEARCH EXPRESSION, ONE TABLE RETRIEVAL}, to T-DELIVER;

3. T-DELIVER passes the first topic, BASIC CONCEPTS, from the sorted set of topics to TUTOR-SUBTOPICS;

4. TUTOR-SUBTOPICS sorts the child topics of BASIC CONCEPTS, and pass the unit topics, CONSTANT, VARIABLE, and EXPRESSION, one by one, to the discussing module by calling procedure T-DISCUSS;

5. T-DELIVER passes the second topic, SEARCH EXPRESSION, from the sorted set of topics to TUTOR-SUBTOPICS;

6. TUTOR-SUBTOPICS sorts the child topics of SEARCH EXPRESSION, and pass the unit topics, SIMPLE SEARCH EXPRESSION and COMPOUND SEARCH EXPRESSION, one by one, to the discussing module by calling procedure T-DISCUSS;

7. T-DELIVER passes the third topic, ONE TABLE RETRIEVAL, from the sorted set of topics to TUTOR-SUBTOPICS;

8. TUTOR-SUBTOPICS sorts the child topics of ONE TABLE RETRIEVAL, and pass the unit topics, UNCONDITIONAL TABLE RETRIEVAL, ONE TABLE SIMPLE RETRIEVAL, and ONE TABLE COMPOUND RETRIEVAL, one by one, to the discussing module by calling procedure T-DISCUSS.

CHAPTER 5

STUDENT KNOWLEDGE REPRESENTATION

The student model (SM) in a Knowledge-Based Tutoring System (KBTS) contains information about the student's mastery of the domain subject (i.e., what materials are known and unknown by the student). Unlike other tutoring systems in which a student model is built on the top of the domain knowledge base DKB, our student model is created based on the topic tree and the precedence relations among the topics. In this chapter, we address three major issues regarding the design and applications of the student model:

1. how a student model is formulated;
2. how a student model is created by the system; and
3. how a student model is used to help the system to select a new topic for a student to study.

5.1 Knowledge Status Tree

In SQL–TUTOR, we use a *knowledge status tree*, which is obtained by attaching the *knowledge status* to the topics of a topic tree, to represent the knowledge status of a student. Therefore, a knowledge status tree is a student model in our system. Formally, a knowledge status tree can be defined as follows:

Definition 13 *A knowledge status tree is a topic tree with each topic T associated with a bipartite $ks(T) = (S_1, S_2)$ called the knowledge status of the topic, where $S_1 \in \{0, 1\}$ is a binary number and $0 \leq S_2 \leq 1$ is a decimal number.* □

The status of a knowledge status tree indicates the system's belief about the mastery degrees of a student to the subject materials. To be more specific, the

student's mastery of the subject materials that are related to a topic T is measure by two factors, the S_1 and S_2 values of topic T. The S_1 value of T, denoted as $S_1(T)$, reflects the student's mastery level to the content of topic T itself. If the system believes that the student has mastered the content of the topic, it assigns $S_1(T) = 1$; otherwise, it assigns $S_1(T) = 0$.

The S_2 value of topic T, denoted as $S_2(T)$, reflects the student's mastery of the subtopics of T. It is defined as $S_2(T) = \frac{(\sum_{i=1}^{j} S_1(T_i))}{j}$, where each $T_i, 1 \leq i \leq j$ and j is the number of the unit subtopics of T, is a unit subtopic of T. This definition of S_2 value has the property that if the student has mastered all the unit subtopics of topic T, then $S_2(T) = 1$; if the student has not mastered any unit subtopic of topic T, then $S_2(T) = 0$; if the student has mastered some unit subtopics of topic T, then $0 < S_2(T) < 1$.

Figure 5.1 shows a knowledge status tree. This tree contains the information about the student's mastery of the topic QT and its subtopics. Some of the information we know from the tree includes:

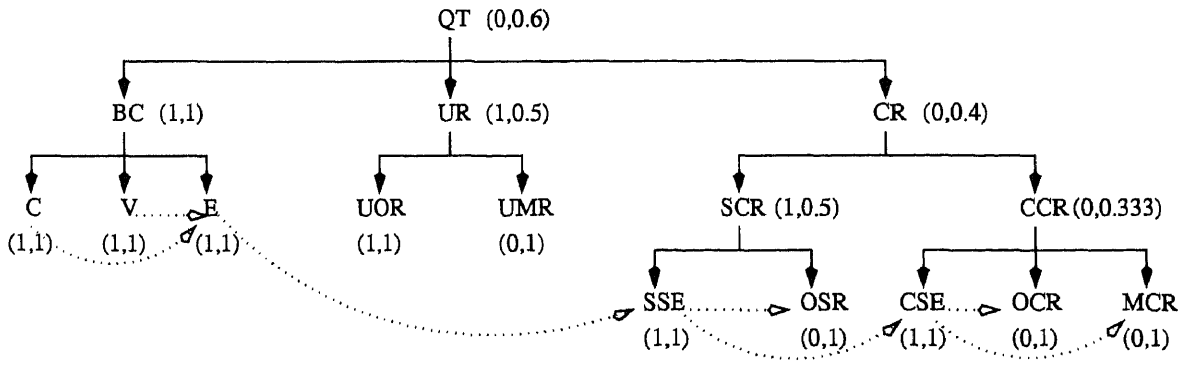


Figure 5.1 A Knowledge Status Tree

- the student has not mastered topic QT ($S_1(\texttt{QT}) = 0$), but mastered some of its subtopics ($S_2(\texttt{QT}) = 0.6$);

- the student has mastered the content of BC (S_1(BC) = 1) and the contents of all its subtopics (S_2(BC) = 1);
- the student has mastered topic UR (S_1(UR) = 1) and part of its subtopics (S_2(UR) = 0.5).

5.2 Creation of Knowledge Status Tree

Given a topic tree, creating a knowledge status tree means to determine the knowledge status for each topic in the topic tree. In order to determine the knowledge status of a topic T, we can associate a set of relevant questions with the topic, pose the questions to the student, and then examine the result of the test. If the student can pass the test associated with T, then assign $S_1(T) = 1$; otherwise, assign $S_1(T) = 0$. Once the S_1 values for all the topics have been determined, we can calculate the S_2 values in a bottom-up fashion, because according to its definition, the S_2 value for a unit topic is always 1 and for a non-unit topic is the average of the S_1 values of its unit subtopics.

However, since asking a student to take a test on every topic of a course is time consuming and it is painful for many students who have to take lots of tests in order to use a tutoring system, this approach is impractical in many cases. Furthermore, a student can easily get frustrated and lose confidence if he fails a large number of tests. (this is especially true for the beginners) Therefore, we want to cut the number of tests taken by the student as many as possible.

Our method for cutting the number of tests is based on the following two principles:

1. if topic T_1 is a precedence of topic T_2 and a student has failed the test on T_1, then the student should also fail the test on T_2, and therefore, it is not necessary for him to take the test on T_2 (or the test on T_2 can be bypassed).

2. if topic T_1 is a precedence of topic T_2 and a student has passed the test on T_2, then the student should also pass the test on T_1, and therefore, it is not necessary for him to take the test on T_1 (or the test on T_1 can be bypassed).

As an application of the principles, consider the topics C and E in the topic tree shown by Figure 5.1. If a student can not pass the test associated with the topic C, then the system can conclude that he can not pass the test associated with the topic E either, because the topic C is a precedence of the topic E. Therefore, the system can assign $S_1(\texttt{E}) = 0$ without asking the student to take the test on the topic E. On the other hand, if the student can pass the test associated with the topic E, then the system can conclude that he should be able to pass the test associated with the topic C also because the topic C is a precedence of the topic E. Therefore, the system can assign $S_1(\texttt{C}) = 1$ without asking the student to take the test on the topic C.

A closer study reveals that the number of the tests which can be cut varies from topics to topics. For instance, for the unit topics in the topic tree shown in Figure 5.1, the topics E, SSE, OSR, CSE, OCR, and MCR can be bypassed if the student fails the test on the topic C, while a failed test on the topic CSE can only cut the tests on the topics OCR and MCR. Similarly, the topics CSE, SSE, E, C, and V can be bypassed if the student can pass the test on the topic OCR or MCR, but a passed test on the topic E can only cut the tests on the topics C and V.

The relationship between a topic T and those topics whose tests can be bypassed due to a test on T can be modeled by a graph called *bypass graph*. In a bypass graph, each node corresponds to a topic, and the nodes are connected by the precedence relations as we do in a TAG. Associated with each node is a pair of numbers, called the F value and the P value of the topic. The F value of a topic T, denoted as $F(T)$, is defined as the number of the tests that can be bypassed if the student has not mastered topic T (i.e., $S_1(T) = 0$), and the P of a topic T, denoted as $P(T)$, is defined as the number of tests that can be bypassed if the student has

mastered the topic T (i.e., $S_1(T) = 1$). Therefore, for the knowledge status tree shown in Figure 5.1, we have $F(\texttt{C}) = 6$; $F(\texttt{CSE}) = 2$; $P(\texttt{OCR}) = 5$; and $P(\texttt{E}) = 2$. In a bypass graph, if there is a path from the topic T_1 to the topic T_2, then T_1 (T_2) can be bypassed by a passed (failed) test on T_2 (T_1).

Figure 5.2 shows the bypass graph for the topics in Figure 5.1.

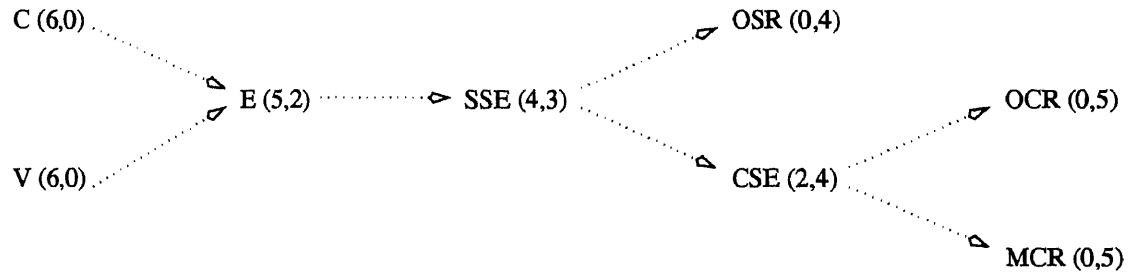


Figure 5.2 The Bypass Graph for the Topics in Figure 5.1

If we use $T_1 \rightarrow T_2$ to denote that topic T_1 should be selected before topic T_2, then the rules used by the system to select the topics can be written as follows:

1. if $\max(F(T_1), P(T_1)) > \max(F(T_2), P(T_2))$, then $T_1 \rightarrow T_2$.
2. if $(F(T_1) + P(T_1)) \geq (F(T_2) + P(T_2))$, then $T_1 \rightarrow T_2$.

When these rules are applied to the bypass graph shown in Figure 5.2, the system will generate this sequence of the topics, (C, V) $\rightarrow$ E $\rightarrow$ (OCR, MCR) $\rightarrow$ SSE $\rightarrow$ CSE $\rightarrow$ OSR, and test these topics by their orders in the sequence. Topics enclosed by a pair of parenthesis can be selected randomly. Randomly selected topics have the same priority.

When the topics are selected by the above rules, their F values and P values are treated equally, that is, F values and P values have the same contributions to the selections. However, this is not the best strategy in some situations. For instance, for a beginner who tends to fails most of the tests instead of passing the tests, the F value of a topic should have more contribution to the topic selection decisions than the corresponding P value, because a failure of a test occurs more often than

a success. This problem can be resolved by assigning the different weights, w_F and w_P, to the F and P values of a topic. Accordingly, we can rewrite the topic selection rules as follows:

1'. if $\mathsf{max}(w_F \cdot F(T_1), w_P \cdot P(T_1)) > \mathsf{max}(w_F \cdot F(T_2), w_P \cdot P(T_2))$ then $T_1 \rightarrow T_2$.

2'. if $(w_F \cdot F(T_1) + w_P \cdot P(T_1)) \geq (w_F \cdot F(T_2) + w_P \cdot P(T_2))$, then $T_1 \rightarrow T_2$.

If $w_F = w_P = 1$, then Rule 1' and Rule 1, Rule 2' and Rule 2 are the same. When $w_F = 1$ and $w_P = 0$, the above selection rules are of the same form and can be replaced by Rule 2' itself.

1'. if $F(T_1) > F(T_2)$, then $T_1 \rightarrow T_2$.

2'. if $F(T_1) \geq F(T_2)$, then $T_1 \rightarrow T_2$.

If the selections of the topics are based on this rule, then they are only dependent on the F values of the topics. This will result in the topic sequence (C, V) $\rightarrow$ E $\rightarrow$ SSE $\rightarrow$ CSE $\rightarrow$ (OSR, OCR, MCR) from the topics shown in Figure 5.2.

5.3 Topic Selection during Planning Phase

In this section, we discuss how to select a new topic for the student to study based on the student model (knowledge status tree). This differs from the discussion of the previous section in which our concern is to cut the number of the tests taken by the student so we can build a student model efficiently.

As we discussed in Section 2.3, the first phase of a communication cycle is the planning phase. During this phase, a new topic is selected for a student to study either by the system (passive mode) or by the student (active mode). If the topic is selected by the system, the first step is to access the student model in the system to find all the unknown topics of the student and the relationships among these unknown topics. This is done by the Planning Module as described below:

1. the Student Status Analyst looks up the knowledge status tree, marks all the topics whose S_1 values are less than 1, and puts them into the Student Unknown Buffer;

2. by consult the curriculum knowledge base (CKB) of the system, the Knowledge Relationship Analyst constructs a Unknown Hierarchy as shown by Figure 5.3 which contains the topic-subtopic and precedence relations among the unknown topics.

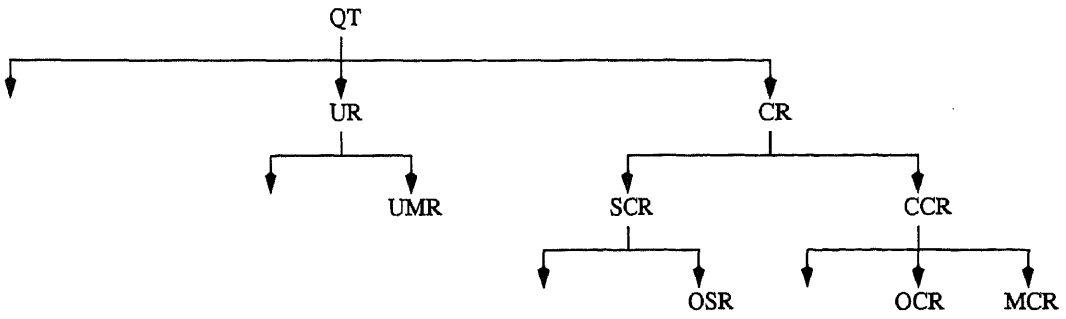


Figure 5.3 A Unknown Hierarchy

3. the Topic Extractor selects a topic from the Unknown Hierarchy and passes it to the Topic Analyst. The selection of the topics are determined by the pedagogical rules stored in the pedagogical knowledge base PKB (see the discussion followed).

4. the Topic Analyst applies the *TAG* operator *STUDY* (which is discussed in Section 4.3.2) to each of the topics passed to generate a learning graph, and sends the learning graph to the Curriculum Delivery.

5. the Curriculum Delivery applies the traversal algorithm discussed in Section 4.4 to traverse the learning graph, and pass the topics currently visited to the Discussing Module.

In the step 3 of the above procedure, a topic is selected from the Unknown Hierarchy by the Topic Extractor based on the pedagogical rules stored in the system's pedagogical knowledge base PKB. Currently, three rules are stored there that are related to this type of selection:

1. if $P(T_1, T_2)$, then $T_1 \rightarrow T_2$. (if topic T_1 is a precedence of topic T_2, T_1 is selected to study before T_2)

2. if $S(T_1, T_2)$, then $T_1 \rightarrow T_2$. (if topic T_1 is a subtopic of topic T_2, T_1 is selected to study before T_2)

3. if $S_2(T_1) \geq S_2(T_2)$, then $T_1 \rightarrow T_2$. (if the student has the better mastery on the subtopics of T_1 than the subtopics of T_2, T_1 is selected to study before T_2)

When these rules are applied to the Unknown Hierarchy shown in Figure 5.3, we get the following outcomes:

1.	UR $\rightarrow$ CR $\rightarrow$ QT	$S(\text{UR}, \text{QT})$
		$S(\text{CR}, \text{QT})$
		$S_2(\text{UR}) > S_2(\text{CR})$
2.	UMR $\rightarrow$ UR	$S(\text{UMR}, \text{UR})$
3.	SCR $\rightarrow$ MCR $\rightarrow$ CR	$S(\text{SCR}, \text{CR})$
		$S(\text{MCR}, \text{CR})$
		$P(\text{SCR}, \text{MCR})$
4.	OSR $\rightarrow$ SR	$S(\text{OSR}, \text{SR})$
5.	(OCR, MCR) $\rightarrow$ CCR	$S(\text{OCR}, \text{CCR})$
		$S(\text{MCR}, \text{CCR})$

Therefore, the sequence of the topics sent to the Discussing Module is UMR $\rightarrow$ UR $\rightarrow$ OSR $\rightarrow$ SCR $\rightarrow$ (OCR, MCR) $\rightarrow$ CCR $\rightarrow$ CR $\rightarrow$ QT.

CHAPTER 6

DECLARATIVE DOMAIN KNOWLEDGE REPRESENTATION

The domain knowledge, which is also called the subject domain, of a knowledge based tutoring system (KBTS) consists of the subject materials from a course. This is the target knowledge that a tutoring system intends to teach its students and can be categorized into either declarative knowledge or procedural knowledge. The *declarative domain knowledge* can be expressed as declarative statements (such as definitions and descriptions of the various domain concepts) and stored as symbolic structures (such as semantic networks and frames) which are accessible by general procedures.

In this chapter, we discuss how to represent the declarative domain knowledge in a KBTS so that

- the subject materials can be presented to the students effectively; and
- various questions raised by the students about the subject materials can be answered efficiently.

The representation of the procedural domain knowledge in a tutoring system will be discussed in Chapter 8.

6.1 Content of a Topic

In SQL–TUTOR, the knowledge about SQL is stored in the domain knowledge base DKB. Recall that in Chapter 3, we have defined a topic T in a tutoring system as a bipartite $T = (N_T, C_T)$, where N_T and C_T are the name and the content of T, respectively. For each topic T in the system, there is a node in the node pool NP which contains a pointer called `TOPIC` pointing to the subject material associated with T.

The content of a topic is also called the *document* of that topic. Formally, a document can be defined as follows:

Definition 14 *The document of a topic* $T = (N_T, C_T)$, *denoted as* $D(T)$, *is the content of the topic, that is,* $D(T) = C_T$. □

In our system, the document of topic T is made of three parts:

1. C_s – a set of domain concepts associated with the topic T;
2. C_x – the context of the topic which describes the relationships among a group of domain concepts; and
3. C_a – a set of of auxiliary materials of the topic T which include examples, working problems (exercises), and quizzes associated with the topic T.

Figure 6.1 shows the template for defining a topic.

<table>
<tr><td colspan="3">NAME (NT)</td></tr>
<tr><td rowspan="5">DOCUMENT
(CT)</td><td rowspan="4">CONCEPT SET
(Cs)</td><td>CONCEPT</td></tr>
<tr><td>CONCEPT</td></tr>
<tr><td>. . . . . .</td></tr>
<tr><td>CONCEPT</td></tr>
<tr><td colspan="2">CONTEXT (Cx)</td></tr>
<tr><td rowspan="3">AUXILIARY
MATERIALS
(Ca)</td><td>EXAMPLES</td></tr>
<tr><td>WORKING PROBLEMS</td></tr>
<tr><td>QUIZZES</td></tr>
</table>

Figure 6.1 The Template for a Topic

A topic T may cover several concepts in its domain and thus contains the subject materials for all of them. These materials are stored in the *concept set* C_s in the document associated with T. For instance, the document of topic SIMPLE SEARCH EXPRESSION in SQL–TUTOR contains the subject materials for three SQL concepts: SIMPLE COMPARISON, RELATIONAL OPERATOR and PRECEDENCE ORDER. We will discuss the design of the concept set of a document in Section 6.2.

A set of domain concepts can be related to one another. The knowledge about the relationships among the domain concepts is represented by the contexts of the documents. The context of a document can be used by a tutoring system to answer the student's questions regarding the relationships among the concepts. We will discuss the design of the context of a document in Section 6.3.

The examples in a document have two applications during a discussing phase:

1. they can be a part of a lecture presented to the student by the system. During a discussing phase, the Domain Knowledge Extractor receives a topic from the planning module, uses the topic as the key to retrieve the document associated with the topic from the domain knowledge base DKB, and puts the various parts of the documents, including the examples, in the Subject Content Buffer to allow the Lecture Generator to generate a lecture (see Figure 2.5);

2. they can also be shown to the student at his request during a discussing phase.

The purpose of associating a set of working problems W_D and a set of quizzes Q_D with a document is to allow the system to be able to pose the appropriate exercises and quizzes to the students during an evaluating phase. When a topic is passed to the evaluating module from the discussing module, the Evaluation Generator will use this topic as the key to retrieve the working problems and quizzes from the DKB and put them in the Problem Buffer (see Figure 2.6). The Problem

Descriptor poses first the working problems to the student for the practice purpose, and then poses the quizzes to test the student performance.

6.2 The Representation of Concept

As we have introduced in the previous section, part of the declarative knowledge in SQL–TUTOR is stored in the concept set and the context of a document. In this section, we first examine the two aspects of the declarative knowledge, then discuss how to represent the concepts in a document.

6.2.1 Two Forms of Declarative Knowledge

Declarative knowledge can be represented by two types of forms. The first type of the forms uses a natural language to describe concepts and their relationships. English, French, German, and Chinese are among the well known natural languages. We call this form of knowledge the *natural form representation* of the knowledge. Examples of this type of representations drawn from algebra, physics, and SQL courses are given below.

> The absolute value of a real number is its distance from zero on the number line. The absolute value is never negative (algebra).
>
> Whenever there is a net force **F** acts on an object, it produces an acceleration **a** in the direction of the force which is proportional to the magnitude of the force and inversely proportional to the mass of the object **m** (physics).
>
> The SELECT statement is used to retrieve information from a database. This is the most frequently used statement in SQL. (SQL)

The second form of the representation uses an abstract language to describe concepts and their relationships. Typical abstract languages include logical expressions, mathematical formula, tables, graphs, and other formal descriptions such as the Backus Normal Form (BNF). We call this form of knowledge the *abstract form representation* of knowledge. For example, Newton's second law and the syntax of

SQL SELECT statement can be expressed by the following mathematical and BNF formula, respectively.

1. $\mathbf{a} = \propto \frac{\mathbf{F}}{m}$; $\mathbf{F} = m\mathbf{a}$;
2. select $< column\_list > \mid *$
   from $< table\_list >$
   where $< search\_condition >$

Since the abstract representation uses the formal notations for describing domain knowledge, it is often necessary to attach natural language explanations to the formal notations to help people understand their meanings. Examples of this type of abstract notations with natural explanations are shown below:

1. $\mathbf{a} = \propto \frac{\mathbf{F}}{m}$; $\mathbf{a}$ is proportional to $\mathbf{F}$ and inversely proportional to m;
2. $\mathbf{F} = m\mathbf{a}$; $\mathbf{F}$ is equal to the product of $\mathbf{a}$ and m.

6.2.2 Components of a Concept

Formally, a concept in the concept set of a document can be defined as follows:

Definition 15 *A concept is a quadruple $C = (C_n, C_d, C_s, C_f)$, where*

1. *C_n is the name of the concept;*
2. *C_d is the description of the concept;*
3. *C_s is the supplement material associated with the concept; and*
4. *C_f is a set of formal notations associated with the concept and their natural explanations.* □

The first element of a concept is its name C_n which must be unique for each concept throughout the system. DATABASE and TABLE are two examples of the concept names in SQL–TUTOR. The second element of a concept is its description C_d, which

is a natural description of the concept (such as those shown in the previous section). The third element of a concept is its supplement material C_s, which provides either additional information about the concept for refining the second element, or another form of natural description for the concept. In the latter case, the supplement part serves as another perspective of the concept. For example, given the following description about the concept `JOIN` in SQL:

> `JOIN` is a relational operation which retrieves data from two or more tables.

We may associate the following supplement to it to refine the description:

> The data retrieved by `JOIN` are constrained by the *join condition*.

We may also associate the following supplement to it to provide another perspective for the concept:

> `JOIN` is a relational operation which retrieves a subset of the Cartesian product of two or more tables.

The fourth element of a concept is a set of formal notations and their natural explanations C_f. For instance, we can associate the following formal notation and natural explanation to relational operation `JOIN`:

$T_1 \bowtie_{<join\ condition>} T_2$	Select all combinations of tables T_1 and T_2 that satisfy the *join condition*

6.2.3 Concept List

In our system, the domain concepts are linked together by the concept lists. The system maintains a linked list (the system concept list) which contains all the concepts in the subject domain. Each topic also has a concept list which is a part of the system concept list and contains all the concepts in the content of the topic. A node in a concept list consists of four fields:

1. LABEL – the label of the topic of which the concept is a content (e.g., QT, BC and CR).

2. CONCEPT – the name of the concept (e.g., SIMPLE COMPARISON, JOIN).

3. NEXT_CONCEPT_IN_SYSTEM – a pointer pointing the next node in the system concept list.

4. NEXT_CONCEPT_IN_DOC – a pointer pointing the next concept which is in the same document as this node.

Figure 6.2 shows a node pool (which is a part of the curriculum knowledge base), a set of topics, and the concept lists associated with the concepts (which are parts of the domain knowledge base).

The following functions are used for accessing information stored in the documents and concept lists.

1. FIND_DOC(C,TAG) – Return a pointer pointing to the document which contains the concept C.

```
FIND_DOC(C,TAG)
{
   for (i=1; i<=TAG.Length; i++) {
      p = TAG.NP[i].TOPIC->Cs;
      tpc = TAG.NP[i].LABEL;
      while (p != NULL && p->TOPIC == tpc)
         if (p->CONCEPT == C)
            return TAG.NP[i].TOPIC;
         else
            p = p->NEXT_CONCEPT_IN_DOC;
   }
   return NULL;
}
```

2. IS_IN_DOC(C,T,TAG) – If concept C is a domain concept defined in the document of topic T, return 1; return 0 otherwise.

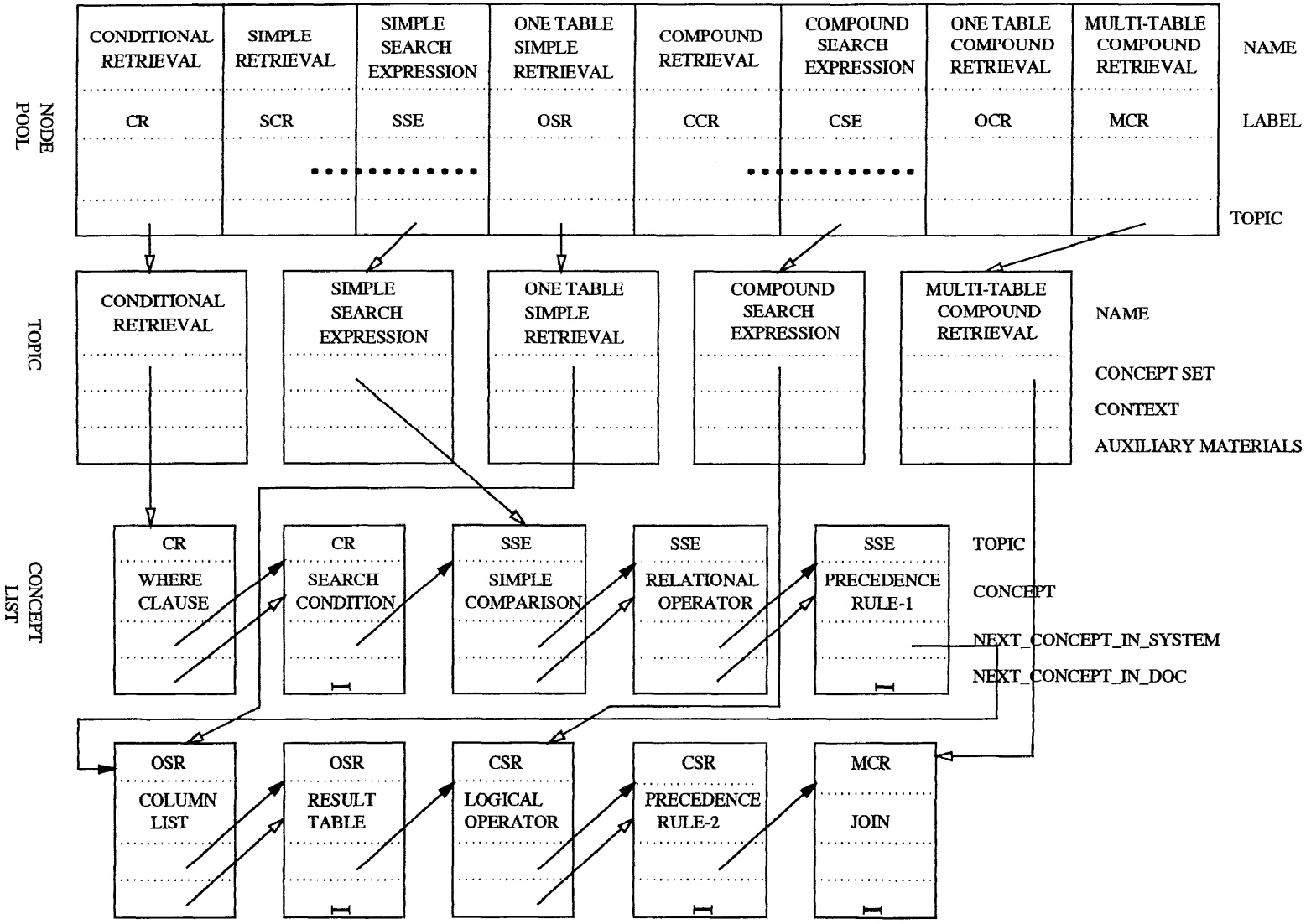


Figure 6.2 Node Pool, Topics and Concept Lists

```
IS_IN_DOC(C,TAG)
\* C is a concept, T is a pointer pointing to a document *\
{
   for (i=1; i<=TAG.Length; i++)     \* find the document of T *\
      if (TAG.NP[i].NAME == T) {
         Tp = TAG.NP[i].TOPIC;
         break;
      }
   Dp = FIND_DOC(C,TAG);      \* find the document containing C *\
   return (Tp==Dp ? 1:0);
}
```

3. IS_DOMAIN_CONCEPT(C,T,TAG) – If concept C is in the domain of topic T, return 1; return 0 otherwise.

```
IS_DOMAIN_CONCEPT(C,T,TAG)
{
   if (IS_IN_DOC(C,T,TAG))
      return 1;
   else {
      let i = Index(T,TAG.NP);
      for (j=1; j<=TAG.Length; j++) {
         T1 = LABEL(j,TAG);
         if (IS_SUBTOPIC(j,i,TAG) && IS_IN_DOC(C,T1,TAG))
            return 1;
      }
   return 0;
}
```

6.3 Context Network

The DESCRIPTION and SUPPLEMENT sections of a document contain the definitions, descriptions, and explanations for the various domain concepts, whereas the CONTEXT section of a document describes the relationships among the domain concepts. The knowledge represented by the context of a document allows a tutoring system to answer various questions raised by the students during a discussing phase. In this section, we discuss the design of the context. In the next chapter, we will

introduce how a tutoring system can use the information in the contexts to answer the student's questions.

We have designed a type of semantic network, called the *Context Network*, to represent the contexts of the documents. A context network consists of a set of nodes and a set of links connecting the nodes. Each node in a context network corresponds to a concept and there are two types of them: subject nodes and prerequisite nodes. A *subject node* (or s-node) corresponds to a concept to be defined by the system. Examples of the subject nodes in SQL–TUTOR include `DATABASE`, `TABLE` and `SQL`. If the concept of an s-node in the context of a document is defined by another document, this s-node is also called a *reference node* (or r-node). A *prerequisite node* (or p-node) in a context network corresponds to a concept that should be known by the student before he takes the course. Examples of prerequisite nodes in SQL–TUTOR course include `DATA`, `INFORMATION` and `SOFTWARE`.

Each link in a context network represents a relationship between the two domain concepts connected by it and has a label describing the relationship. We use `link-label`(N_i, N_j, C_x) to denote that there is a link with label `link-label` connecting nodes N_i and N_j in the context network C_x. There are two types of links in a context network:

- relational link (r-link) – connecting a subject (source) and an object (target);
- descriptive link (d-link) – connecting an object (source) and a prepositional object (target).

As an example, consider the following description about the relationship between the concepts database and database management system: *A database management system manages data in a database.* This description can be represented by the context network in Figure 6.3.[1] In this context network, there are

[1]In defining a context network, each node can have at most one d-link leaving from it.

two s-nodes: DATABASE MANAGEMENT SYSTEM and DATABASE; one p-node: DATA; one relational link: MANAGE(DATABASE MANAGEMENT SYSTEM, DATA); and one descriptive link: IN(DATA, DATABASE).

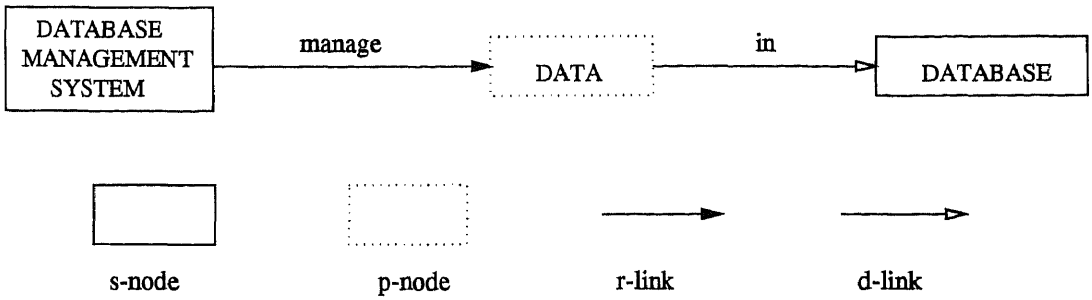


Figure 6.3 An Example of Context

Table 6.1 contains a list of the most frequently used relationships by context networks. It also contains examples explaining these relationships.[2]

Table 6.1 Commonly Used Relations by Contexts

RELATIONS	EXAMPLES
kind_of	SQL is a kind of Data Definition Language.
synonym_of	DDL is a synonym of Data Definition Language.
type_of	A relational database is a type of database.
part_of	A table body is a part of table.
set_of	A table body is a set of rows.
instance_of	100 is an instance of integer.
has_a	A column has a domain.

[2]Although both kind_of and type_of relationships can be used to describe the containment (is_a) relationship, they have a difference: if we can enumerate all the subclasses of an object, then we use type_of relationships; otherwise, we use kind_of relationship.

6.4 Examples of Documents

In this section, we present the sample documents from SQL–TUTOR.[3] The first document is associated with the topic RELATIONAL DATA MODEL whose context network includes six subject nodes (RELATIONAL DATABASE, TABLE, ROW, COLUMN, TABLE NAME, TABLE HEADING, TABLE BODY, COLUMN NAME, DOMAIN) and two reference nodes (CHARACTER STRING and DATA TYPE). The second document is associated with the topic RELATIONAL OPERATIONS whose context network includes four subject nodes (RELATIONAL OPERATION, SELECTION, PROJECTION, JOIN), one prerequisite node (DATA), and three reference nodes (TABLE, ROW, COLUMN).

6.4.1 Example 1: RELATIONAL DATA MODEL

CONCEPT SET (C_s)

CONCEPT 1

Name: RELATIONAL DATABASE

Description: A RELATIONAL DATABASE is a collection of two dimensional structures called tables.

Supplement: A relational database contains data organized in files called tables.

Notation: [ ]

CONCEPT 2

Name: TABLE

Description: A TABLE contains information about objects and relationships among objects and has a unique name within the database.

Supplement: The data in a table is organized in rows and columns. The structure of a table is determined by a table schema. Each table in a database usually contains a different kind of information.

Notation: [ ]

CONCEPT 3

[3]The designs of the examples inside the documents will be discussed in Chapter 8.

Name: ROW

Description: A ROW of a table contains a single item of information about one object.

Supplement: A row contains information about one of the objects that the table describes.

CONCEPT 4

Name: COLUMN

Description: A COLUMN of a table contains information about one property of the objects.

Supplement: Each column has a name which is used to refer to the column in the table.

Notations: [ ]

Context (C_x)[4]

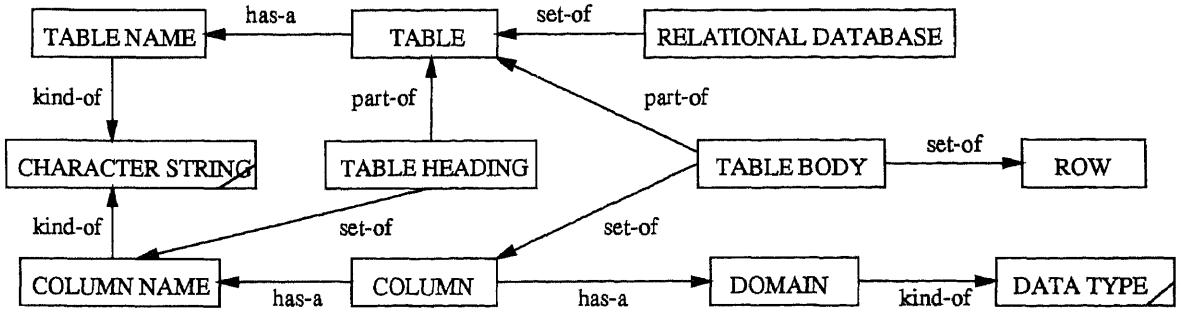


Figure 6.4 Context of RELATIONAL DATA MODEL

6.4.2 Example 2: RELATIONAL OPERATIONS

Concept Set (C_s)

CONCEPT 1

Name: RELATIONAL OPERATION

Description: A RELATIONAL OPERATION is used to retrieve data from tables in a database.

[4]A r-node in a context network is represented by a solid rectangle with a slash symbol at the lower right corner.

Supplement: The relational data model supports three basic relational operations, SELECTION, PROJECTION, and JOIN on tables.

CONCEPT 2

Name: `SELECTION`

Description: `SELECTION` is an operation which picks out only certain *rows* from a table.

Supplement: The selection operation selects a subset of the rows from a table that satisfy the selection condition.

Notation:

$\sigma_{<selection\ condition>}(T)$	Select rows from table T that satisfy the *selection condition*

CONCEPT 3

Name: `PROJECTION`

Description: `PROJECTION` is an operation which picks out only certain *columns* from a table.

Supplement: The projection operation selects one or more columns from a table.

Notation:

$\pi_{<column\ list>}(T)$	Produce a new table with only columns from the *column list*

CONCEPT 4

Name: `JOIN`

Description: `JOIN` is an operation which produces all combinations from two tables that satisfy the join condition.

Supplement: To join two tables T_1 and T_2, you must make sure that one column in each table has the same type of data.

Notation:

$T_1 \bowtie_{<join\ condition>} T_2$	Select all combinations from tables T_1 and T_2 that satisfy the *join condition*

Context (C_x):

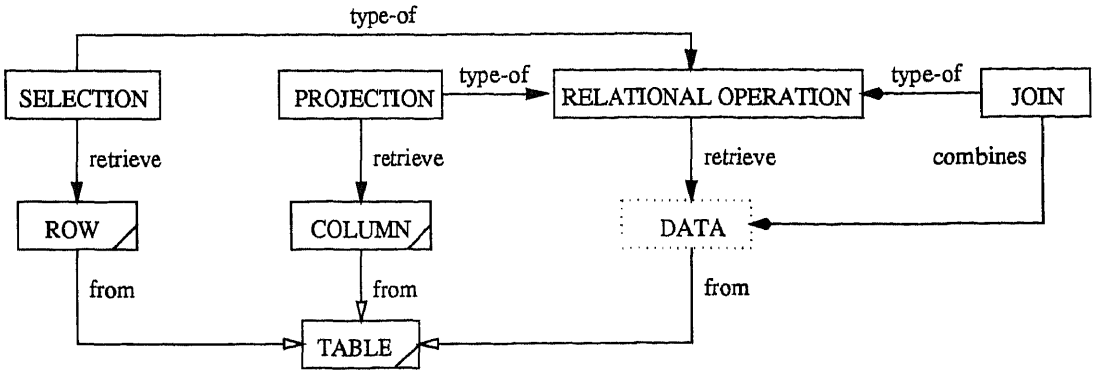


Figure 6.5 Context of RELATIONAL OPERATION

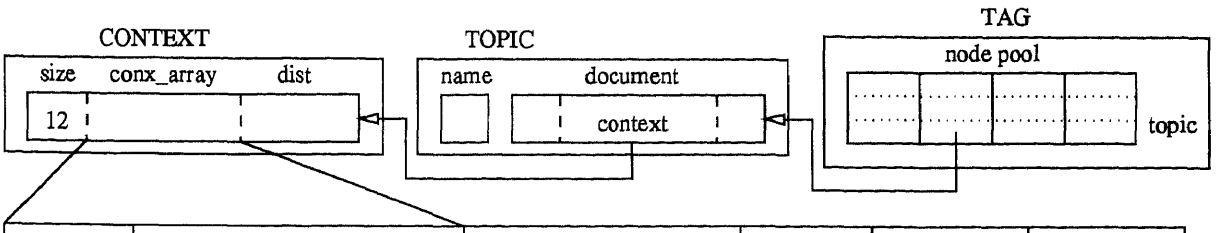
6.4.3 Internal Representation of Context

The context of a document is represented by a record consisting of an integer (SIZE) and two arrays (CONX_ARRAY and DIST). The CONX_ARRAY which is the internal representation of the context network whose size (the number of records) is defined by the value of SIZE. Each element in the CONX_ARRAY is a record with six fields:

1. LINK_NAME – the label of the link;
2. SOURCE – the name of the source concept;
3. TARGET – the name of the target concept;
4. LINK_TYPE – the type of the link (r-link or d-link);
5. SOURCE_TYPE – the type of the source concept (s-node, r-node or p-node); and
6. TARGET_TYPE – the type of the target concept (s-node, r-node or p-node).

Figure 6.6 shows the CONX_ARRAY associated with topic RELATIONAL DATA MODEL (cf. Figure 6.4). It also depicts the relationships among the node pool of a *TAG*, the topics of the *TAG* and the contexts of the topics.

Function IS_IN_CONTEXT(C,Cx) is used to determine if concept C corresponds to a node in the context Cx.



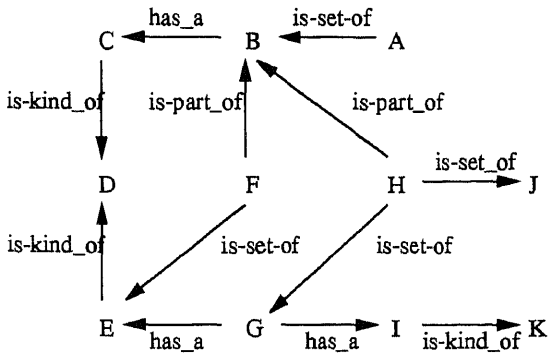
LINK NAME	SOURCE	TARGET	LINK TYPE	SOURCE TYPE	TARGET TYPE
set-of	RELATIONAL DATABASE	TABLE	r-link	s-node	s-node
set-of	TABLE BODY	ROW	r-link	s-node	s-node
set-of	TABLE BODY	COLUMN	r-link	s-node	s-node
has-a	TABLE	TABLE-NAME	r-link	s-node	s-node
has-a	COLUMN	COLUMN-NAME	r-link	s-node	s-node
has-a	COLUMN	DOMAIN	r-link	s-node	s-node
kind-of	TABLE NAME	CHARACTER STRING	r-link	s-node	r-node
kind-of	COLUMN NAME	CHARACTER STRING	r-link	s-node	s-node
kind-of	DOMAIN	DATA TYPE	r-link	s-node	r-node
part-of	TABLE HEADING	TABLE	r-link	s-node	s-node
part-of	TABLE BODY	TABLE	r-link	s-node	s-node
set-of	TABLE HEADING	COLUMN NAME	r-link	s-node	s-node

Figure 6.6 A Context and its Context Array

```
IS_IN_CONTEXT(C,Cx)
{
   for (i=1; i<=Cx.size; i++)
      if (C == Cx.CONX_ARRAY[i].SOURCE || C == Cx.CONX_ARRAY[i].TARGET)
         return 1;
   return 0;
}
```

The third item of a context record is an array called the *distance matrix*, which contains the distances between each pair of the nodes in the context. The distance between the two nodes is defined as the length of the shortest path between them in

the context. Figure 6.7 shows a context network and the distance array associated with it.



	A	B	C	D	E	F	G	H	I	J	K
A	0	1	2	3	3	2	3	2	4	3	5
B	1	0	1	2	2	1	2	1	3	2	4
C	2	1	0	1	2	2	3	2	4	3	5
D	3	2	1	0	1	2	2	3	3	4	4
E	3	2	2	1	0	1	1	2	2	3	3
F	2	1	2	2	1	0	2	2	3	3	4
G	3	2	3	2	1	2	0	1	2	1	3
H	2	1	2	3	2	2	1	0	2	1	3
I	4	3	4	3	2	3	2	2	0	3	1
J	3	2	3	4	3	3	1	1	3	0	4
K	5	4	5	4	3	4	3	3	1	4	0

Figure 6.7 A Context and its Distance Matrix

Given a context network Cx, algorithm CONSTRUCT_DIST creates the distance matrix for the context.

```
CONSTRUCT_DIST(Cx)
\* Create the distance array for context network Cx *\
{
   for (each node C in Cx) {
      FLAG(C) = 0;
      DIST[C,C] = 0;
   }

   for (each node C in Cx) {
      open = {C};       \* open is a list of nodes to be explored *\
      FLAG(C) = 1;
      while (open is not empty) {
         X = the first node in open;
         remove X from open;
         for (each node Y adjacent to X in Cx)
            if (FLAG(Y) = 0) {
               Cx.DIST[Y,C] = Cx.DIST[X,C] + 1;
```

```
                FLAG(Y) = 1;
                open = open + {Y};    \* add Y to the end of open *\
              }
        }
    }
}
```

6.5 The Design of Sample Database

We have designed a sample database called COMPANY in SQL–TUTOR from which various examples illustrating database concepts can be derived. The sample database consists of two components: an Enhanced System Catalog (ESC) and five sample tables. The five sample tables are: EMP, DEPT, PROJ, DEPT_LOC, and WORK_ON. A set of sample data for COMPANY is shown in Figure 6.8.

EMP

John	Smith	123-456	09-JAN-55	234-567	5
Frank	Wong	234-567	08-DEC-45	456-789	5
Alicia	Zelaya	345-678	19-JUL-58	234-567	4
James	Borg	456-789	11-NOV-27	-	1

DEPT

5	Research	234-567	3366
4	Administration	234-567	1256
1	Headquarters	456-789	0234

PROJ

1	ProductX	Bellaire	5
2	ProductY	Sugarland	5
3	Reorganization	Houston	1
4	Computerization	Stafford	4

DEP_LOC

1	Houston
4	Stafford
5	Bellaire
5	Sugarland

WORK_ON

123-456	1	32
234-567	2	15
345-678	3	25
234-567	1	20

Figure 6.8 Sample Database COMPANY

The enhanced system catalog in SQL–TUTOR is used to store the information about the sample tables. It contains:

1. the name and content of each table.

2. the name, content, and structure of each column of each table.

3. the addresses of the definition and reference rules[5] associated with the table and its columns.

EMP

CONTENT	EMPLOYEE					
COLUMN NAME	NAME		ID	BDATE	SID	DNO
	LNAME	FNAME				
COLUMN MEANING	Employee's Last Name	Employee's First Name	Employee's ID Number	Employee's Birthday	Employee Supervisor's ID	Employee's Department Number

DEPT

CONTENT	DEPARTMENT			
COLUMN NAME	DNUMBER	DNAME	MID	PHONE
COLUMN MEANING	Department Number	Department Name	Department Manager's ID	Department Phone Number

PROJ

CONTENT	PROJECT			
COLUMN NAME	PNUMBER	DNAME	DNUM	LOCATION
COLUMN MEANING	Project Number	Department Name	Department Number	Project Location

DEPT_LOC

CONTENT	DEPARTMENT LOCATION	
COLUMN NAME	DNUM	LOCATION
COLUMN MEANING	Department Number	Department Location

WORK_ON

CONTENT	EMPLOYEE AND PROJECT		
COLUMN NAME	ID	DNAME	DNUM
COLUMN MEANING	Employee's ID Number	Project Number	Employee's Working hour

Figure 6.9 SQL–TUTOR Enhanced System Catalog

The system catalog in a relational database usually holds information on what privileges each individual user has, what tables exist in the database, what columns are there in each table, what type of data can be stored in each column, etc.

[5]The discussion of the definition and reference rules can be found in Chapter 7

Compared with this type of system catalogs, the enhanced system catalog in SQL–TUTOR contains one more type of information, the content (or semantic meaning) of each table in the database and each column in a table. Figure 6.9 shows the ESC in SQL–TUTOR.

6.6 Summary of the Chapter

In this chapter, we discussed how to represent the declarative knowledge in a KBTS. Our representation schema has the following features:

1. the declarative knowledge base consists of two parts: a set of documents associated with the topics and the sample database in a system;

2. each document consists of a set of domain concepts, a context, and a group of auxiliary materials (examples, exercises and quizzes);

3. each concept in the concept set of a document is made of a name, a description, an optional supplement and an optional formal notation;

4. the context of a document contains the information about the relationships among the domain concepts.

5. the enhanced system catalog of the sample database can be used by the system to answer questions from a student, and the sample tables can be used to derive various examples.

CHAPTER 7

SQL–TUTOR QUESTION ANSWERING MECHANISM

In the previous chapter, we discussed how to represent the domain knowledge in a knowledge based tutoring system. One strength of our domain knowledge representation schema comes from its inference capability. With this capability, a tutoring system can search for the knowledge encoded implicitly in a domain knowledge base and allow a student to ask various types of questions about the subject matter. In this chapter, we discuss in detail how SQL–TUTOR answers the questions raised by the student based on the information stored in the domain knowledge base.

7.1 Types of Questions and Answers

Recall that during a discussing phase, after a student asks a question, the Student Question Analyst converts the question into an internal format and puts it in the Question Description Buffer (see Figure 2.5). A student is restricted to ask SQL–TUTOR any of the following four types of questions about a topic:

1. what-is type. For example, "What is a table?"
2. what-notation type. For example, "What is the notation of the JOIN operator?"
3. what-relationship type. For example, "What is the relationship between a relational database and a table?"[1]
4. what-meaning type. For example, "What is the meaning of the table EMP?"

The internal format of a question created by the Student Question Analyst contains two parts: the type of the question and the concept(s) involved in

[1] A yes/no type of question can be converted into a what–relationship type of question. For example, instead of asking "Is a relational database a collection of tables?" we can ask, "What is the relationship between a relational database and a table?"

the question. Table 7.1 shows some examples of questions and their internal representations.[2] If Q is the question stored in the Question Buffer, then we use Q.TYPE to refer to the type of the question, Q.OBJ1, Q.OBJ2 and Q.OBJ3 to refer to the first, second and third objects in the question, respectively.

Table 7.1 Question and Their Internal Representations

QUESTIONS	INTERNAL FORMATS			
	TYPE	OBJECT 1	OBJECT 2	OBJECT 3
WHAT IS A TABLE?	1	TABLE		
WHAT IS THE FORMAT OF JOIN OPERATOR?	2	JOIN OPERATOR		
WHAT IS THE RELATIONSHIP BETWEEN A RELATIONAL DATABASE AND A TABLE?	3	RELATIONAL DATABASE	TABLE	
WHAT IS THE MEANING OF TABLE EMP?	4.1	EMP		
WHAT IS THE MEANING OF COLUMN FNAME IN TABLE EMP?	4.2	FNAME	EMP	
WHAT IS THE MEANING OF TUPLE "5, Research, 234-567,3366" IN TABLE DEPT?	4.3	"5, Research, 234-567, 3366"	DEPT	
WHAT IS THE MEANING OF DATA "234-567" IN COLUMN MID OF TABLE DEPT?	4.4	"234-567"	MID	DEPT

The formats of the answer will vary depending on the types of the questions. Therefore, there are four of them.

7.1.1 Answer Type 1

The answer to a what-is type of question contains the DESCRIPTION and SUPPLEMENT parts of the concept. For example, if the student asks, "What is a table?" then from the sample document RELATIONAL DATA MODEL described in the previous chapter, the answer from SQL–TUTOR will be:

[2] A what-meaning type of question may ask for the meaning of a table, a column of a table, a tuple of a table, or a data stored in a column of a table. Therefore, there are four sub-types of this type of questions.

A TABLE *contains information about objects and relationships among objects and has a unique name within the database.* (DESCRIPTION)

The data in a table is organized in rows and columns. The structure of a table is determined by a table schema. Each table in a database usually contains a different type of information. (SUPPLEMENT)

7.1.2 Answer Type 2

The answer to a what-notation type of question contains the NOTATION part of the concept. For example, if the student asks, "What is the notation for JOIN operator?" then from the sample document RELATIONAL DATA MODEL described in the previous chapter, the answer from SQL–TUTOR will be:

$T_1 \bowtie_{<join\ condition>} T_2$	Select all combinations from tables T_1 and T_2 that satisfy the *join condition*

7.1.3 Answer Type 3

The answer to a what-relationship type of question describes the relationship between two domain concepts. For instance, if the question raised by a student is, "What is the relationship between a relational database and a table?" the system will answer:

RELATIONAL DATABASE is-set-of TABLES.

7.1.4 Answer Type 4

The answer to a what-meaning type of question describes the semantic meaning of an object (a table, column, tuple, or an atomic data) in the sample database. For example, if the student asks, "What is the meaning of column FNAME in table EMP?" the system will answer:

Column FNAME of table EMP contains information about **employee first name**.

Table 7.2 shows the formats and examples of the answers created by SQL–TUTOR. The first column of the table lists the type of an answer. The second

column lists the kinds of objects contained in the answer. The last column shows some examples of the answers.

Table 7.2 Answers and Their Internal Representations

TYPE	OBJECT	EXAMPLE
1	CONCEPT NAME	(1, TABLE)
2	CONCEPT NAME	(2, JOIN OPERATOR)
3	SOLUTION PATH	(3, [TABLE,part-of(b),TABLE BODY,set-of(f), ROW]
4.1	TABLE NAME	(4.1, EMP)
4.2	TABLE NAME, COLUMN NAME	(4.2, EMP, FNAME)
4.3	TUPLE, TABLE NAME	(4.3, "5, Research, 234-567, 3366", DEPT)
4.4	ATOMIC DATA, COLUMN NAME, TABLE NAME	(4.4, "234-567", MID, DEPT)

The Solution Buffer SBUF in a system is defined as a two-field record: TYPE and OPTR, where TYPE stores the type of an answer and OPTR is a pointer pointing to a linked list which contains the objects in the answer. Each node in the linked list consists of four fields:

1. OBJ – the name of the object (either a concept and a link);
2. LINK_TYPE – the type of a link (either r (r-link) or d (d-link));
3. LINK_DIR – the direction of a link (either f (forward) or b (backward));
4. NEXT – a pointer to the next object in the answer.

Figure 7.1 shows the Solution Buffers corresponding to the third and the last answers given in Table 7.2.

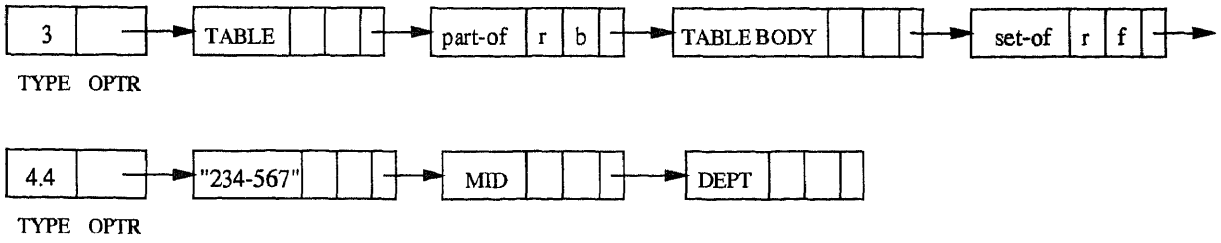


Figure 7.1 The Solution Buffer

7.2 Problem Solver

The Discussing Module of SQL-TUTOR contains a sub-module called the Problem Solver which governs the construction of an answer for the given question. Once a question has been added to the Question Description Buffer, the Problem Solver will be invoked. It constructs an answer and puts the answer into the Solution Buffer. Figure 7.2 shows the relationship between the Problem Solver and the knowledge bases in the system.

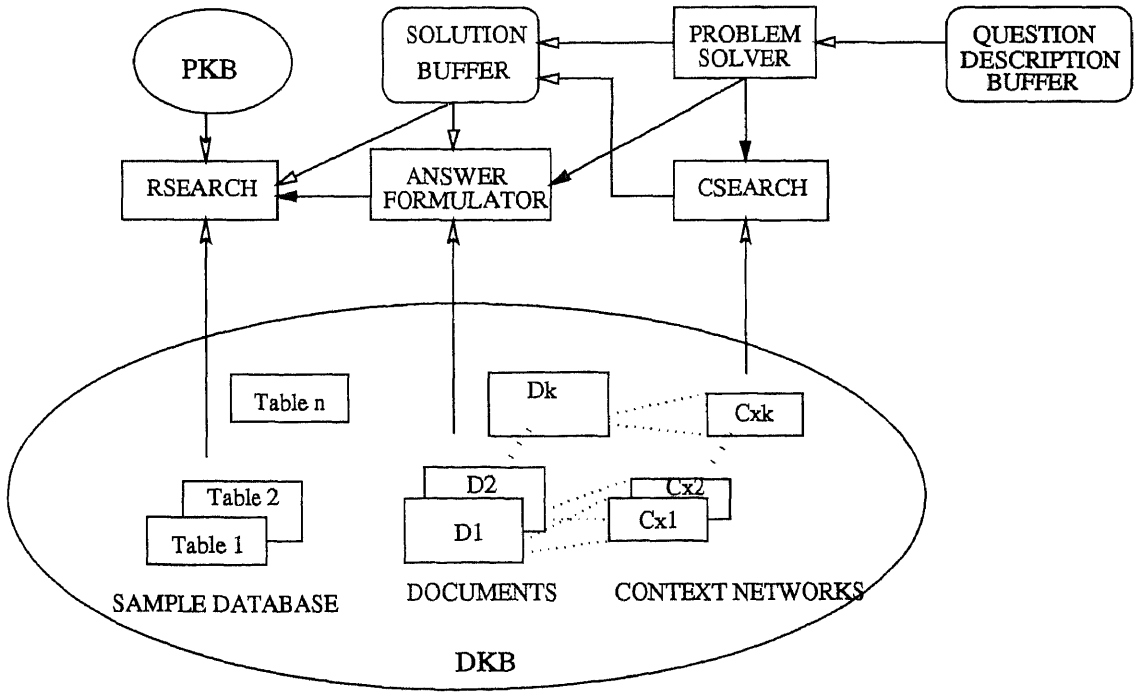


Figure 7.2 Problem Solver and Knowledge Bases

The Problem Solver calls two other procedures during the construction of an answer. The CSEARCH procedure is used to answer a what-relationship type of question. It searches the context networks for the relationship between two domain, constructs a solution path consisting of the node names and link labels from the context networks, and puts the solution path into the Solution Buffer. We will discuss the procedure CSEARCH in detail in Section 7.3.

The ANSWER FORMULATOR is in charge of translating the information in the Solution Buffer into the corresponding natural language description. In the process of this translation, it may need to retrieve the various parts (DESCRIPTION, SUPPLEMENT, NOTATION) from the documents by searching the set of document definitions in the system. If the question is of what-meaning type of question, the ANSWER-FORMULATOR invokes the RSEARCH procedure to search the pedagogical rules stored in the pedagogical knowledge base PKB, find the applicable rules to the given question, and execute the rules to in order ro generate an answer. We will discuss the design of the ANSWER FORMULATOR and RSEARCH in detail in Sections 7.4 and 7.5, respectively.

The following algorithm describes the Problem Solver in SQL–TUTOR.

```
PSOLVER(TAG)
\* create a solution path for the question Q in the Question *\
\* Description Buffer, put it into the Solution Buffer, call *\
\* ANSWER-FORMULATOR to display the solution to the student  *\
{
    SBUF.TYPE = Q.TYPE;
    C1 = Q.OBJ1;
    switch (Q.TYPE) {
    case (1):
    case (2):
       SBUF.OPTR->OBJ = C1;
       SBUF.OPTR->NEXT = NULL;
       break;
    case (3):
       C2 = Q.OBJ2;
       CSEARCH(C1,C2,TAG,SBUF);
```

```
      break;
   otherwise:
      SBUF.OPTR->OBJ = C1;
      if (Q.TYPE == 4.1) {
         SBUF.OPTR->NEXT = NULL;
      else {
         C2 = Q.OBJ2;
         SBUF.OPTR->NEXT->OBJ = C2;
         if (Q.TYPE == 4.4)
            SBUF.OPTR->NEXT->NEXT->OBJ = Q.OBJ3;
            SBUF.OPTR->NEXT->NEXT->NEXT = NULL;
         else {
            SBUF.OPTR->NEXT->NEXT = NULL;
         }
      }
   }   /* switch */
   ANSWER-FORMULATOR(TAG,SBUF);
}
```

7.3 The Context Searching Algorithm CSEARCH

The Context Searching algorithm CSEARCH takes a question of the form "What is the relationship between nodes C1 and C2?" and tries to create a solution path from the starting node C1 to the goal node C2. If such a path can be found (that is, there is a context containing both C1 and C2), CSEARCH calls another procedure CSEARCH1 to construct a solution path and put it into the Solution Buffer. The following is a description of the CSEARCH.

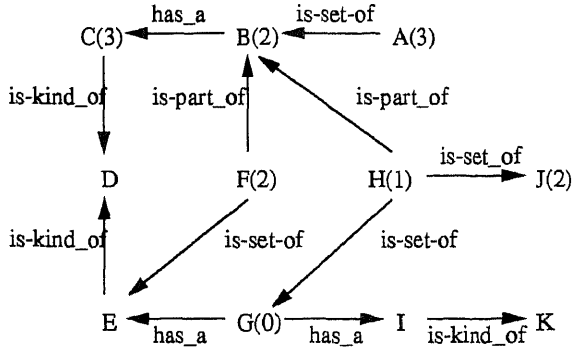
```
CSEARCH(C1,C2,TAG,SBUF)
\* Construct a solution path from C1 to C2 in TAG and puts it *\
\* in the Solution Buffer SBUF *\
{
   D1 = FIND_DOC(C1,TAG);          \* Find the document  *\
   Cx1 = D1->CONTEXT;              \* and context of C1  *\
   \* if C2 is in the same context as C1 *\
   if (IS_IN_CONTEXT(C2,D1))
      \* construct the solution path *\
      SPath = CSEARCH1(Cx1,C1,C2);
   else {  \* C2 is not in the context Cx1 *\
      D2 = FIND_DOC(C2,TAG);        \* Find the document *\
```

```
      Cx2 = D2->CONTEXT;            \* and context of C2 *\
      \* if C1 is in the same context as C2 *\
      if (IS_IN_CONTEXT(C1,D2))
         \* construct the solution path *\
         SPath = CSEARCH(Cx2,C1,C2);
      else  \* C1 and C2 are not in the same context *\
         SPath = NULL;
   }
   SBUF = SPath;
}
```

The solution path constructed by CSEARCH1 is made of the names of nodes and the labels as well as the directions of the links connecting these nodes. We denote such a path by [C1,L1(d),N1,L2(d),N2,.....,Nk,Lk(d),C2], where each N_i is a node name and each L_i is a link label. The direction of a link is either f (forward link) or b (backward link). The CSEARCH1 uses the distance matrix associated with the context to find the shortest path between C1 and C2. It starts from C1, selects an adjacent node C of C1 such that C has the shortest distance to C2, adds C and the link label between C1 and C to the partial solution path, and then starts the search from C. The algorithm can be described as follows:

```
CSEARCH1(Cx,C1,C2)
\* Search the context network Cx to construct a solution path *\
\* from C1 to C2 *\
{
   MinDist = 10000;
   for (each link in Cx of form Ln(C1,C') or Ln(C',C1))
      if (C' == C2)
         if (Ln(C1,C'))    \* find forward link from C1 to C2 *\
            return ([C1,Ln(f),C2]);
         else              \* find backward link from C1 to C2 *\
            return ([C1,Ln(b),C2]);
      else
         if (DIST[C',C2] < MinDist) {
            C = C';
            MinDist = DIST[C',C2]
         }

   SPath = CSEARCH1(C,C2,Cx);
```



	A	B	C	D	E	F	G	H	I	J	K
A	0	1	2	3	3	2	3	2	4	3	5
B	1	0	1	2	2	1	2	1	3	2	4
C	2	1	0	1	2	2	3	2	4	3	5
D	3	2	1	0	1	2	2	3	3	4	4
E	3	2	2	1	0	1	1	2	2	3	3
F	2	1	2	2	1	0	2	2	3	3	4
G	3	2	3	2	1	2	0	1	2	1	3
H	2	1	2	3	2	2	1	0	2	1	3
I	4	3	4	3	2	3	2	2	0	3	1
J	3	2	3	4	3	3	1	1	3	0	4
K	5	4	5	4	3	4	3	3	1	4	0

Solution Path: [A,is-set-of(f),B,is-part-of(b),H,is-set-of(f),G]

Figure 7.3 A Search Graph and a Solution Path

```
    if (Ln(C1,C))
       return (append(C1,Ln(f),SPath));
    else
       return (append(C1,Ln(b),SPath));
}
```

As an example, Figure 7.3 shows a search graph generated from the CSEARCH. The shortest solution path from node A to node G is marked by the bold links. The numbers besides the nodes are their distances to the goal node G.

7.4 Answer Formulator

After the Problem Solver puts a solution path in the Solution Buffer, the Answer Formulator will translate the answer from the internal format (solution path) into its corresponding natural language description so the student can read and understand it. Besides the information in the Solution Buffer, the Answer Formulator also uses the DESCRIPTION, SUPPLEMENT and NOTATION parts of a concept in a document. It may also invoke the RSEARCH procedure to formulate an answer if the question

is of a what-meaning question. The following algorithm describes the **ANSWER** FORMULATOR.

```
ANSWER-FORMULATOR(TAG,SBUF)
\* formulate an answer from the internal format stored in the *\
\* Buffer SBUF and display the answer *\
{
   TYPE = SBUF.TYPE;
   C = SBUF.OPTR->OBJ
   D = FIND_DOC(C,TAG);
   switch (TYPE) {
      case (1): {
         \* display the description of concept C in document D *\
         DISPLAY_DESC (D,C);
         \* display the supplement of concept C in document D *\
         DISPLAY_SUPP (D,C);
         break; }
      case (2): {
         \* display the notation of concept C in document D *\
         DISPLAY_NOTA (D,C);
         break; }
      case (3): {
         \* translate a solution path to an answer *\
         DISPLAY_PATH(C,Cp);
         break; }
      otherwise:
         \* search the meaning of an object in sample database *\
         RSEARCH(SBUF);
      }
}
```

Procedure DISPLAY_PATH translates a solution path stored in the Solution Buffer into an answer and display the answer.

```
DISPLAY_PATH(C,Cp)
\* C is a concept in the Solution Buffer, Cp is a pointer *\
\* pointing to the link connecting C and the next node in *\
\* the solution buffer *\
{
   while (Cp != NULL) {
      C' = Cp->NEXT->OBJ;          \* C' is the next object *\
      if (Cp->LINK_DIR = 'f')      \* if a forward link *\
```

```
        display "C Cp->LINK_NAME Cp->NEXT->OBJ";
      else                          \* if a backward link
        display "C' Cp->LINK_NAME Cp->NEXT-OBJ";

      if (Cp->LINK_TYPE = 'r') {    \* if a r-link  *\
        C = C';
        Cp = Cp->NEXT->NEXT;          \* get next link *\
      }
   }
}
```

For instance, if the solution path stored in the Solution Buffer is [TABLE, is-part-of(b), TABLE BODY, is-set-of(f), ROW], then the answer displayed by the DISPLAY_PATH is

TABLE BODY is-part-of TABLE; TABLE BODY is-set-of ROW.

7.5 Find the Meaning of an Object: RSEARCH

A student can ask the meanings of four types of objects in the sample database. These objects are:

1. a *table* in the sample database (question type 4.1). For example, "What is the meaning of the table EMP?" In this case, SQL–TUTOR will response with the kind of information stored in the table (the semantics of the table) by looking up the Enhanced System Catalog ESC. Therefore, the answer to the above question will be

   "Table EMP contains INFORMATION ABOUT AN EMPLOYEE."

2. a *column* in a sample table (question type 4.2). For example, "What is the meaning of the column MID in the table EMP?" In this case, SQL–TUTOR will response with the kind of information stored in the column (the semantics of the column) by looking up the ESC. Therefore, the answer for the question above from the system will be

"Column MID of table DEPT contains information about manager's ID number."

3. a *tuple* of a table in the sample database (question type 4.3). In this case, SQL–TUTOR will explain the meaning of the tuple to the student by applying the definition semantic rule (see Section 7.5.1) of the table.

4. a *sample data* of a column in a table (question type 4.4). In this case, SQL–TUTOR will response with the semantic meaning of the data item. There are two possibilities based on the type of the column which contains the selected data:

   (a) If the column is an ordinary one, SQL–TUTOR will just tell the student that piece of data is an instance of the property represented by the column. For example, "John is an instance of first name." and "Research is an instance of department name".

   (b) If the column refers to another column in a different table, SQL–TUTOR will apply some reference rule (see Section 7.5.2) to find the "deeper meaning" of that piece of data.

7.5.1 Definition Rule

In the relational database model, the relationships among object attributes are modeled through columns of tables. One column can relate to other columns in the same table or from different tables. In SQL–TUTOR, the relationships among columns are governed by two kinds of semantic rules in the pedagogical knowledge base: definition rules and reference rules. A *definition rule* is a rule which describes the meaning of a tuple in one table, whereas a *reference rule* is a rule which explains the semantic meaning of a sample data of a column which refers to another column in a different table.

For each table in the sample database COMPANY, there is one definition rule associated with it. The left-hand side of a definition rule consists of the table name and a list of parameters, whereas the right-hand side of rule is a template for generating answers. The number of parameters in the left-hand side should be equal to the number of columns in the table. When a tuple from the table has be selected by the student, the associated rule will get *fired* (activated) and the parameters in the left-hand side of the rule will be instantiated by the data from the selected tuple. This instantiation will then be passed to the right-hand of the rule to cerate an answer.

For example, suppose the definition rule associated with the table DEPT is

IF DEPT(DNUMBER, DNAME, MID, PHONE)
THEN There is a department such that
the department number is DNUMBER,
the department name is DNAME,
the manager's ID number is MID,
and the department phone number is PHONE.

If a student has selected the first tuple from the table DEPT ("5, Research, 234-567, 3366"), then the following instantiations will take place:

- 5 → DNUMBER
- Research → DNAME
- 234-567 → MID
- 3366 → PHONE

Finally, the answer generated by the system will be:

There is a department such that the department number is 5, the department name is Research, the manager's ID number is 234-567, and the department phone number is 3366.

7.5.2 Reference Rule

A reference rule is used to generate an answer to a student if he has asked a question about the semantic meaning of a data in a column referring to another column (foreign key) in a different table. The column which contains the selected data is called the *referring* column. Like a definition rule, the left-hand side of a reference rule may contain parameters which can be instantiated by the data in the sample database. This instantiation of parameters will then pass to the right-hand side of the rule to generate an answer.

For example, in the sample database COMPANY, each department has a manager and the column MID records its manager's ID number, which should be one of the ID numbers stored in the column ID of the table EMP. This relationship between MID of DEPT and ID of EMP can be described by the following reference rule (Notice that the referring column is XID):

```
IF      DEPT(_,DNAME,XID,_), EMP(LNAME,FNAME,XID,_,_,_)
THEN  Employee FNAME LNAME is the manager of DNAME department
```

We have used the underscore _ , a Prolog-like notation, in a parameter list to denote a *dummy parameter*, whose actual value will not be included in an answer and thus has no affect on the final format of the answer.

In Figure 7.4, we can think there exists a matching link between column MID of table DEPT and column ID of table EMP. If a student has selected data "234-567" in column MID from table DEPT, then MID will have value 234-567. This value will then pass to ID column in table EMP because of the matching link between them. Then the system applies EMP(LNAME,FNAME,234-567,_,_) and DEPT(_,DNAME,234-567,_) to match those tuples in the sample database, and yields the following instantiations:

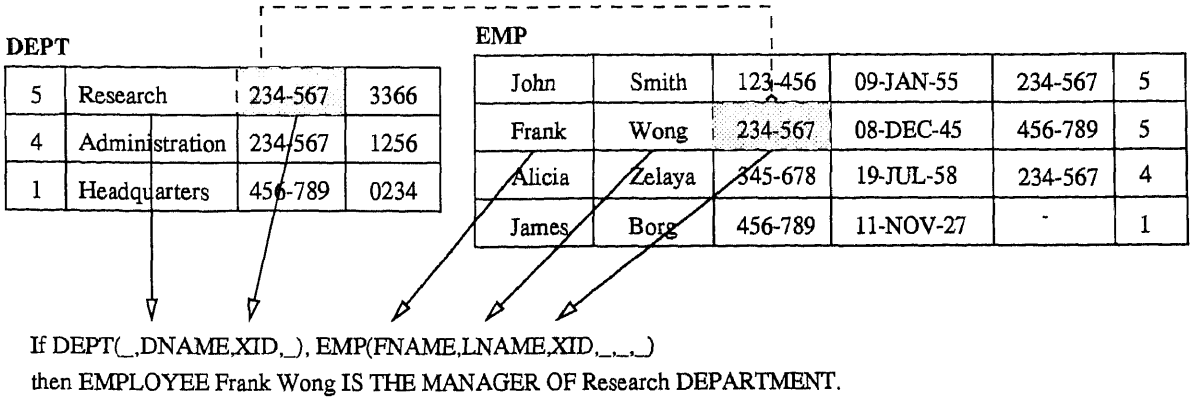


Figure 7.4 The Application of a Reference Rule.

1. {\tt Research} $\rightarrow$ {\tt Dname}

2. {\tt Wong} $\rightarrow$ {\tt LNAME}

3. {\tt Frank} $\rightarrow$ {\tt FNAME}

Therefore, the answer generated is "Employee **Frank Wong** is the manager of Research department."

7.5.3 RSEARCH Procedure

The procedure RSEARCH converts the internal format of an answer stored in the Solution Buffer into its natural language description and displays it to the student. During this process, it may apply definition rule or reference rule to generate the answer.

```
RSEARCH(SBUF)
{
  \* TBL is the table name in answer *\
  TBL = SBUF.OPTR->OBJ;
  switch (SBUF->TYPE) {
    case (4.1) {
      search the Enhanced System Catalog ESC for table TBL;
      T_CONT = Content of TBL found in ESC;
      display "Table TBL contains information about T_CONT";
      break; }
```

```
    case (4.2) {
      \* CLM is the column name *\
      CLM = SBUF.OPTR->NEXT->OBJ;
      search the Enhanced System Catalog ESC for TBL and CLM;
      C_CONT = Content of CLM in table TBL found in ESC;
      display "Column CLM in Table TBL contains information
                    about C_CONT";
      break; }
    case (4.3) {
      \* TPL is a tuple from sample database *\
      TPL = SBUF.OPTR->NEXT->OBJ;
      apply the definition rule for table TBL to tuple TPL;
      display the right-hand side of the definition rule;
      break; }
    case (4.4) {
      \* DATA is a data from sample database *\
      DATA = SBUF.OPTR->NEXT->OBJ;
      CLM = referring column in table TBL;
      if (CLM is a foreign key in table T') {
         apply the reference rule for TBL and T' to DATA;
         display the right-hand side of the reference rule; }
      else
         display  "DATA is an instance of Column CLM in
                              Table TBL"; }
  }
}
```

CHAPTER 8

REPRESENTING THE KNOWLEDGE OF WRITING SQL QUERIES

The previous two chapters focus on the specification of instruction – the DESCRIPTION, SUPPLEMENT, NOTATIONS and CONTEXT aspects of a document. In this chapter, we discuss how to teach problem solving skills to a student, using writing SQL queries as example. Since each problem solving skill is taught through various concrete examples, this approach is called *Teaching By The Example.*

8.1 The Design of an Example

Many researchers suggest that the most obvious difference between an expert and a novice is that the expert, who has more knowledge than the novice, also organizes the knowledge more effectively than the novice does. Because of this knowledge organization, an expert can rapidly evoke the particular items relevant to the problem at hand [16, 37]. Many research has been done to determine the organization in which the expert's knowledge is held in the long-term memory. Evidences from the areas of understanding story, learning text editor, and learning programming language support the assumption that the expert has built up large libraries of stereotypical solutions (or canned solutions) for problems as well as strategies for coordinating and decomposing them [3, 62]. This kind of stereotypical solutions for problems is called the *chunk* of knowledge [37]. It has been characterized in terms of goals and plans, and represented by frames [63] and rules [33, 52].

In SQL–TUTOR, the knowledge chunk for writing SQL queries is stored in the example associated with a document (remember that an example is an element of a document). Each example is used to guide the students to solve one type of the domain problems and contains

- a set of text descriptions of the problem;

- a graphical representation (Semantic Query Graph) of the domain problem which can be used to derive solutions for the problem and generate meaningful feedbacks to the student.

In the following sections, we will discuss different parts of an example and their applications in a tutoring process.

8.1.1 Problem Description

The set of problem descriptions associated with one example describe the same problem from different perspectives. By providing descriptions of different levels, they help the student understand a problem correctly at the first place. As an example, consider the following three descriptions for one problem:

1. *Find out the birthday and the social security number of 'John Smith'. (Description D_1)*

2. *Retrieve the birthday and the social security number from the employee whose name is 'John Smith'. (Description D_2)*

3. *Retrieve the birthday and the social security number from the employee whose first name is 'John' and last name is 'Smith'. (Description D_3)*

Compared with the other two descriptions, description D_1 is the most general one. Such a description is independent to the design of a specific database schema. For the most people, it is also the easiest one to understand. Description D_2 is more specific than D_1, because it states one fact (i.e., 'John Smith' is an employee) which was not mentioned explicitly in D_1. Description D_3 is the most specific one, because it adds another fact (i.e., 'John' is a first name and 'Smith' is a last name) to its description. In fact, D_3 is very close to an SQL query. Such a description depends on the fact that the `NAME` field of an employee is composite which consists of two subfields, first name and last name.

In SQL–TUTOR, we typically associate three or four descriptions to each problem in an example, organize and display them from the most general one to the most specific one to the students, and hope that the students can understand the problem by studying these descriptions.

8.1.2 Semantic Query Graph

A Semantic Query Graph (*SQG*) is a graph which represents a domain problem by its nodes and links. Each node in an *SQG* corresponds to one object in an SQL query and consists of three fields:

- index: a unique integer identifying the node. We usually use the index of a node, `index`, to denote the node.
- name: the name of the node, which can be the name of a table, a column, a constant, or an operator in the SQL query. Sometimes, we also use the index and name pair `(index,name)` to denote a node.
- type: the type of the node. There are four of them:
  1. *table node (T-node)*: if the node represents a table in the sample database;
  2. *field node (F-node)*: if the node represents a column in a table;
  3. *constant node (C-node)*: if the node represents an SQL constant; and
  4. *operator node (O-node)*: if the node represents an SQL operator.

An O-node in an *SQG* can be further categorized into a relational O-node or a logical O-node. A relational O-node corresponds to an SQL relational operator (`=`, `!=`, `>`, `<`, `>=`, and `<=`), whereas a logical O-node corresponds to an SQL logical operator (`and` and `or`).

A link in an *SQG* can be either an *attribute link*, if it connects a T-node and a F-node, or an *operation link*, if it connects two O-nodes, or one O-node and one F-node. An *SQG* should satisfy the following properties:

- Every F-node has one outgoing link to either a T-node or an O-node, and/or one incoming link from a T-node.
- Every C-node has at least one outgoing link to an O-node.
- Every O-node has two incoming links. For a relational O-node, one link is from an F-node, and another link is from either an F-node or a C-node. For a logical O-node, both links are from O-nodes.
- Every but one O-node has one outgoing link to a logical O-node.

Figure 8.1 shows an *SQG* which consists of one T-node `(3,EMP)`; four F-nodes `(1,BDATE)`, `(2,SSN)`, `(5,LNAME)`, `(6,FNAME)`; two C-nodes `(4,Smith)`, `(7,John)`; and three O-nodes `(8,=)`, `(9,=)`, `(10,AND)`. Note that the last O-node, `(10,AND)`, is the one without outgoing link.

The Semantic Query Graph *SQG* can play an important role for teaching a problem solving skill. It is used by the system for three purposes:

1. generating SQL queries which can solve the given problem associated with the concepts obtained in the topic and display these queries to the students for their reference purposes;
2. creating constructive feedbacks to the students if they have difficulties to solve the given problem and need system helps;
3. examining the queries written by the students and evaluating their performance during their problem solving procedures.

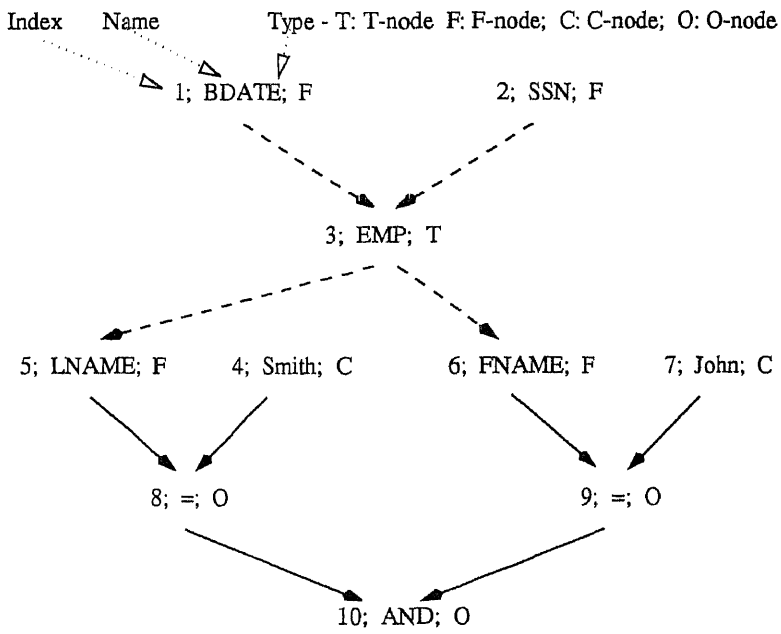


Figure 8.1 A Semantic Query Graph

In the following sections, we will discuss how a tutoring system can fulfill these tasks based on the *SQG*.

8.2 From Semantic Query Graph to SQL Queries

The algorithm used by SQL-TUTOR to convert an *SQG* into the equivalent SQL queries is called `QUERY-BUILDER` (`QB`), which generates `SELECT-FROM-WHERE` form of SQL queries. In other words, each query generated by `QB` has the format

```
SELECT column-list
FROM   table-list
WHERE  search-expression.
```

The `QB` calls procedure `CONS-S-EXP` to construct the search expression in the `WHERE` clause of an SQL query. The `CONS-S-EXP` starts at the operator node, say `(I,O)`, with two incoming nodes, `(I1,N1)` and `(I2,N2)`, and without any outgoing

link. If one of its incoming nodes is an F-node, then the search expression N1 O N2 will be returned. Otherwise, two search expressions, S1 and S2, starting from (I1,N1) and (I2,N2), respectively, will be constructed first and the search expression S1 O S2 will be returned by the algorithm. The following is a description of the procedure:

```
CONS-S-EXP(SQG,(I,O))
\* construct a search expression for node (I,O) from SQG *\
\* (I,O) has two incoming nodes and no outgoing node  *\
{
   (I1,N1) = the first incoming node of (I,O);
   (I2,N2) = the second incoming node of (I,O);
   if ((I1,O1) is an f-node || (I2,O2) is an f-node)
      return (N1,O,N2);
   else
      return (CONS-S-EXP(SQG,(I1,N1)) O CONS-S-EXP(SQG,(I2,N2)));
}
```

For example, given the *SQG* shown in Figure 8.1, CONSTRUCT-S-EXP builds the search expression staring at the O-node (10,AND). Since this node is connected to two other O-nodes (8,=) and (9,=), we will first construct two search expressions S1 and S2, starting at (8,=) and (9,=), respectively, and then return S1 AND S2. Because the node (8,=) connects an F-node (4,LNAME) and a C-node (5,Smith), the algorithm will return LNAME = 'Smith' as the value of S1. Similarly, the search expression S2, FNAME = 'John', will be returned from the node (9,=). Finally, the search expression of the whole query will be

```
LNAME = 'Smith' AND FNAME = 'John'
```

Now, we can describe QUERY-BUILDER as follows:

```
QUERY-BUILDER(SQG)
\* Given an SQG, return the SQL queries derived from it *\
{
   columns = [ ];    tables = [ ];
   foreach (F-node (I,N) in SQG)
      if ({\tt (I,N)} has only one outgoing links)
```

```
        add (I,O) to columns;
    foreach (T-node(I,T) in SQG)
        add (I,T) to tables;
    find the O-node (I,O) without outgoing links;
    search-exp = CONS-S-EXP(SQG,(I,O));
    return ("SELECT",columns,"FROM",tables,"WHERE",search-exp);
}
```

When we apply the above algorithm to the *SQG* of Figure 8.1, we may obtain the following results:

1. columns = {BDATE, SSN}
2. tables = {EMP}
3. search-exp = {LNAME = 'Smith' AND FNAME = 'John'}

Therefore, the SQL query generated is

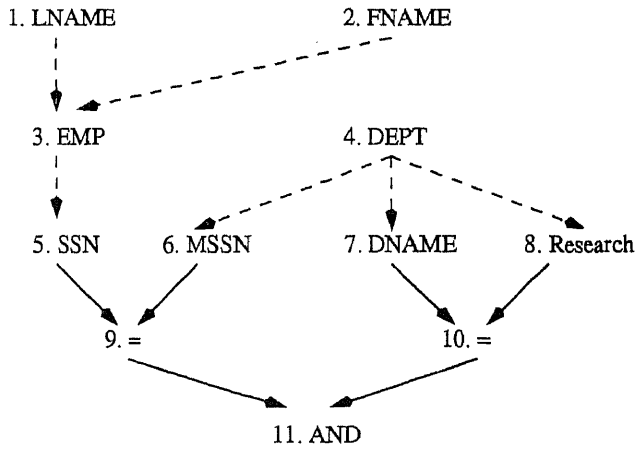
```
SELECT BDATE, SSN
FROM EMP
WHERE LNAME = 'Smith' AND FNAME = 'John'
```

An interesting feature of these two algorithms is their nondeterministic property. F-nodes and T-nodes are selected randomly by the **QUERY-BUILDER** and added into the column list and table list. Similarly, the **CONSTRUCT-S-EXP** also randomly picks up the two incoming nodes of the O-node and constructs the corresponding search expressions. Because of these random selections, several SQL queries can be generated from one single *SQG*.

8.2.1 An Example

In this section, we give a complete example which can be a member of example set of the skill two-table-retrieval.

- PROBLEM DESCRIPTIONS
  1. Find the name of the manager of the department 'Research'.
  2. Find the name of the manager of the department whose name is 'Research'.
  3. Find the name of the employee who is the manager of the department whose name is 'Research'.
  4. Find the first name and the last name of the employee who is the manager of the department whose name is 'Research'.
- SEMANTIC QUERY GRAPH:



8.3 Annotated Semantic Query Graph

The feedbacks from a tutoring system to its students can be very helpful for the students to understand why a solution is right or wrong. The feedbacks from SQL–TUTOR can answer two types of questions:

1. why an object (a table, field, a constant, or an operator) should be included in a query; and

2. why an object should not be included in a query.

In order to answer these questions about a query, we first create an Annotated Semantic Query Graph (*ASQG*) by attaching the semantic meanings to each node in a Semantic Query Graph, and then construct an answer based on the semantic

meanings found in an *ASQG*. This section focuses on how to generate an *ASQG*, whereas the next section discusses how to use the construction rules and feedback templates to form an answer.

Given a Semantic Query Graph *SQG*, an *ASQG* is generated by a three step procedure as follows:

1. For each T-node and F-node in the *SQG*, find its semantic meaning by looking up the Enhanced System Catalog (ESC). For example, the semantic meaning of LNAME, *last name of an employee*, can be found in table EMP from the ESC.

2. For each C-node in the *SQG*,

   (a) find the column in the Sample data Base (SDB) which contains this constant;

   (b) find that column and its corresponding semantic meaning from the ESC;

   (c) define its semantics to be "instance of *the semantics of the column*".

   As an example, consider constant 'Smith'. Since we can find it from column LNAME in the sample table EMP and the semantic meaning of LNAME (**last name**) from the ESC, the semantic meaning of 'Smith' is "instance of the *last name of an employee*".[1]

3. For each O-node, use the propagation algorithm described below to find out its semantic meaning.

The propagation algorithm PROP_SEMANTICS *calculates* the semantic meaning of an O-node by looking up and combining the semantics of its two incoming nodes. When the algorithm needs to find out the semantics of an operator, it will check the document unit which discusses the concept of this operator. This document unit

[1] As a consequence, a constant can be used as a sample data only in one column in the SDB.

should be listed in the prerequisite list of the topic. The algorithm can be described as follows:

```
PROP_SEMANTIC(SQG)
\* Given an SQG whose F-nodes and C-nodes are already associated *\
\* with their semantic meanings, calculate the semantic meanings *\
\* of the O-nodes in the SQG *\
{
   mark all F-nodes and C-nodes in SQG;
   while (there is a unmarked O-node (I,O)) {
      select a unmarked O-node such that both of its two
      incoming nodes have been marked;
      call the two incoming nodes (I1,N1) and (I2,N2);
      S1 = the semantics of (I1,N1);
      S2 = the semantics of (I2,N2);
      SO = be the semantics of the operator O;
      associate (S1 S S2) with the node (I,O) as its semantics;
      mark the node (I,O);
   }
}
```

Figure 8.2 shows the *ASQG* obtained by attaching semantic meanings to the *SQG* shown in Figure 8.2.1.

8.4 Construction Rule and Feedback Template

SQL–TUTOR uses its construction rules and feedback templates to create the feedbacks to a student when the student has asked a question about an SQL query. A *feedback template* is a predefined pattern for a type of system responses, which may contain some variable parameters. A *construction rule* specifies under what situations a feedback template should be used to construct a feedback and how to pass the parameters to the feedback template. The construction rules and feedback templates are stored as pairs in SQL–TUTOR. There are fives pairs of them:

- F-RULE and F-TEMPLATE
- T-RULE and T-TEMPLATE

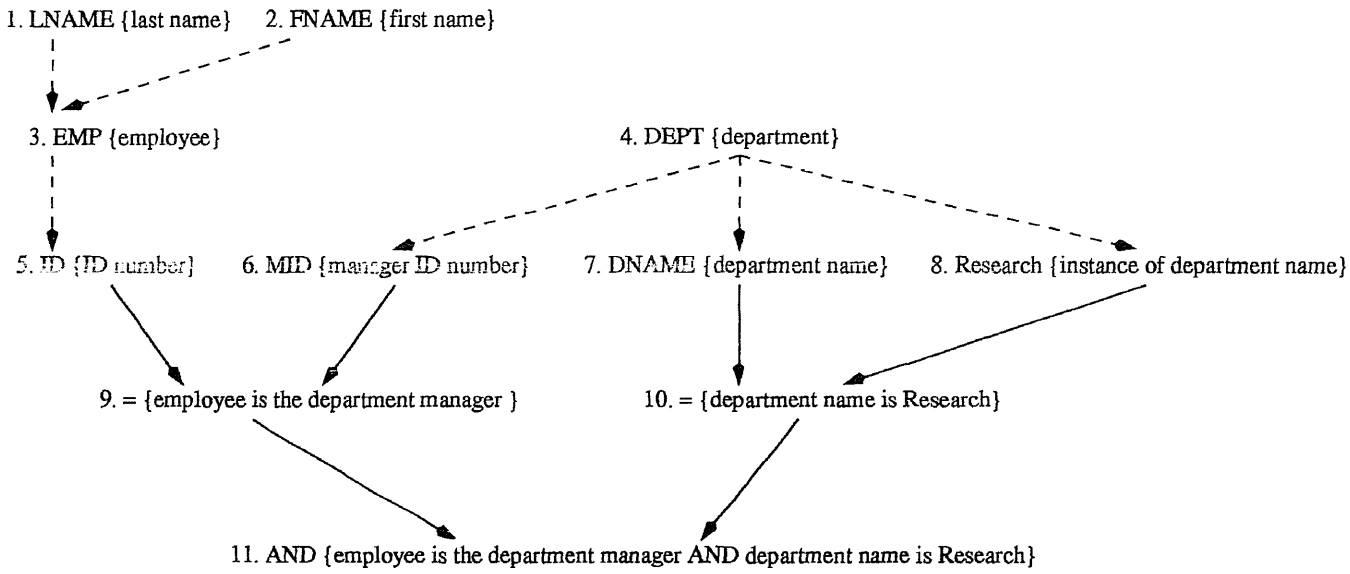


Figure 8.2 An Annotated Semantic Query Graph

- C-RULE and C-TEMPLATE
- O-RULE(R) and O-TEMPLATE(R)
- O-RULE(L) and O-TEMPLATE(L)

Two retrieval functions, NAME and SEM, can be placed in a template to retrieve the name and semantic meaning of a given node from an *ASQG*. The following sections discuss the construction rule/feedback template pairs in detail.

8.4.1 Notations

We use the following notations to describe a construction rule and feedback template:

- T-node(N): node N is a T-node
- F-node(N): node N is an F-node
- C-node(N): node N is a C-node
- O-node(N): node N is an O-node
- RO-node(N): node N is a relational O-node
- LO-node(N): node N is a logical O-node
- from(I,J): there is a link from node I to node J in the *ASQG*;
- ask(I): node I is selected by the student to ask the question.

8.4.2 F-RULE and F-TEMPLATE

The F-RULE will be invoked to construct a feedback if a student has asked a question about an F-node in an SQL query. It passes actual values to the F-TEMPLATE. This rule has two formats:

IF ask(F) and F-node(F) and from(F,T) and T-node(T)

THEN apply F-TEMPLATE(F,T) to construct a feedback.

IF ask(F) and F-node(F) and from(T,F) and T-node(T)
THEN apply F-TEMPLATE(F,T) to construct a feedback.

The first rule can be interpreted as follows: *if* F *is the node that the student asks about, and* F *is an F-node, and there is a link from* F *to* T*, and* T *is a T-node, then pass* F *and* T *to the* F-TEMPLATE *to construct a feedback.* The F-TEMPLATE(F,T) is defined as

> Because the SEM(F) of a(n) SEM(T) is represented by the column NAME(F) in table NAME(T), we include NAME(F) in the query.

For example, given the *ASQG* shown in Figure 8.2 and the F-node selected by the student is (1,LNAME), the F-RULE will be invoked and it will pass (1,LNAME) and (3,EMP) to the F-TEMPLATE. Thus, the two parameters of F-TEMPLATE, F and T, will get values (1,LNAME) and (3,EMP), respectively. Furthermore, the functions NAME and SEM will return the following results:

1. NAME(F) = NAME(1,LNAME) = LNAME
2. NAME(T) = NAME(3,EMP) = EMP
3. SEM(F) = SEM(1,LNAME) = last name
4. SEM(T) = SEM(3,EMP) = employee

Therefore,the feedback constructed from the template will be

> Because the last name of an employee is represented by the column LNAME in table EMP, we include LNAME in the query.

8.4.3 T-RULE and T-TEMPLATE

The T-RULE will be invoked to construct a feedback if a student has asked a question about a T-node in a query. It passes actual values to the T-TEMPLATE. This rule has the format

IF ask(T) and T-node(T), from(F_j,T) ($j = 1, \ldots, m$),
from(T,F_k) ($k = m+1, \ldots, n$) and F-node(F_i) ($i = 1, \ldots, m, \ldots, n$)
THEN apply T-TEMPLATE(T,F_1, ..., F_n) to construct a feedback.

The T-TEMPLATE(T,F_1, ... ,F_n) is defined as

> Because the information about SEM(F_1), ..., SEM(F_n) of a(n) SEM(T) is stored in table NAME(T), we include table NAME(T) in the query.

For example, given the *ASQG* shown in Figure 8.2 and the selected T-node (3,EMP), the T-RULE will be invoked and it will pass (3,EMP), as well as (1,LNAME), (2,FNAME), and (5,SSN) to the T-TEMPLATE. This will result in the following feedback:

> Because the information about last name, first name, and social security number of a(n) employee is stored in table EMP, we include table EMP in the query.

8.4.4 C-RULE and C-TEMPLATE

The C-RULE will be invoked to construct a feedback if a student has asked a question about a C-node in a query. It passes actual values to the C-TEMPLATE. This rule has the format

IF ask(C) and from(C,O) and from(F,O) and from(T,F),
C-node(C) and F-node(F) and O-node(O) and T-node(T)
THEN apply C-TEMPLATE(C,F,O,T) to construct a feedback.

The C-TEMPLATE(C,F,O,T) is defined as

> Because we want to compare the SEM(F) of a(n) SEM(T) with SEM(C) to see if the SEM(F) SEM(O) SEM(C).

For example, given the *ASQG* shown in Figure 8.2 and the selected C-node (8,Research), the C-RULE will be invoked and it will pass (8,Research), (7,DNAME), (10,=), and (4,DEPT) to the C-TEMPLATE. This will result in the following feedback:

> Because we want to compare the department name of a(n) department with Research to see if the department name is Research.

8.4.5 RO-RULE and RO-TEMPLATE

The RO-RULE will be invoked to construct a feedback if a student has asked a question about a relational O-node in a query. It passes actual values to either the RO-TEMPLATE or C-TEMPLATE. This rule has two formats:

IF ask(O) and from(F_1,O) and from(F_2,O) and from(T_1,F_1)
and from(T_2,F_2) and RO-node(O) and F-node(F_1)
and F-node(F_2) and T-node(T_1) and T-node(T_2)
THEN apply RO-TEMPLATE(O,F_1,F_2,T_1,T_2) to construct a feedback.

IF ask(C) and from(F,O) and from(T,O) and from(T,F)
and RO-node(O) and F-node(F) and C-node(C) and T-node(T_1)
THEN apply C-TEMPLATE(C,F,O,T) to construct a feedback.

The RO-TEMPLATE(O,F_1,F_2,T_1,T_2) is defined as

> Because we want to compare the SEM(F_1) of a(n) SEM(T_1) with SEM(F_2) of a(n) SEM(T_2) to see if SEM(O).

For example, given the *ASQG* shown in Figure 8.2 and the selected RO-node (9,=), the RO-RULE will be invoked and it will pass (9,=), (5,ID), (6,MSSN), (3,EMP), and (4,DEPT) to the RO-TEMPLATE. This will result in the following feedback:

Because we want to compare the ID number of a(n) employee with the manager ID number of a(n) department to see if the ID number is manager ID number.

8.4.6 LO-RULE and LO-TEMPLATE

The LO-RULE will be invoked to construct a feedback if a student has asked a question about a logical O-node in a query. It passes actual values to the LO-TEMPLATE. The LO-RULE has the format:

IF ask(O) and from(I_1,O) and from(I_2,O) and LO-node(O)

THEN apply LO-TEMPLATE(O,I_1,I_2) to construct a feedback.

The LO-TEMPLATE(O) is defined as

Because we want to see if SEM(I_1) SEM(O) SEM(I_2).

For example, given the *ASQG* shown in Figure 8.2 and the selected LO-node (11,=), the LO-RULE will be invoked and it will pass (11,=) to the LO-TEMPLATE. This will result in the following feedback:

Because we want to see if employee is the department manager and department name is research.

CHAPTER 9

CONCLUSIONS

In this chapter, we will present some related work in the control for knowledge based tutoring systems and curriculum knowledge representation, summarize the contributions of this research, and consider future directions.

9.1 Related Work

In the physical world, a plan is a description for a sequence of actions that, if followed, will change the situations so as to achieve a desired goal. Many KBTS researchers view a tutoring process as a planning problem [5, 45, 46, 50], where the goal is to have the student learn some domain knowledge. Instructional planners are designed to decide what to do next at each point in an instructional situation. Murray [45, 46] built a blackboard-based dynamic planner which can generate different plans for delivering lessons customized to each individual student. These lesson plans can be revised during tutoring process in response to student problem solving performance and modifications to the student model. The actions in the instructional plan are procedures that control the text, highlighting, and animation displayed to the student. In contrast, the planning module in our system is used to analyze the relationships among related topics and generate learning graphs for the student. The instructional actions are governed by the planning, discussing, evaluating and remedying modules.

The planning approach adopted by SCENT [5] separates an instructional planner into two components: a content planner and a delivery planner. The content planner is responsible for planning the content of a knowledge communication session and the delivery planner decides how to present the content. The planning activity in SCENT centers around the instructional goals such as "have the

student learn recursion". These instructional goals relate to one another in a variety of ways. For example, "have the student learn recursion" is a *part of* "have the student learn LISP programming". The instructional goals and their relationships are embodied in an Goal Knowledge Base (IGKB). An instructional plan consists of a sequence of instructional goals in the IGKB. In SQL–TUTOR, instructional goals (Tgoals) are implicitly embodied in a Topic Association Graph (TAG). The orders of selecting the topics to be tutored are determined by their connections in a learning graph.

The Domain Expert in ExperTutor [24] is constructed from a Goal/Task Hierarchy, which is a lattice of lesson components ordered by an epistemic priority relation. The higher goals in the hierarchy need more expertise than the lower goals. The student has to study and complete all subgoals before he/she can complete a goal successfully. The system applies a "left first depth first" traversal algorithm to generate default instructional sequences. An author can include several rule bases within each node of the hierarchy to provide alternative teaching styles, further examples, remediation, tests, and determine the next action from the system.

Lesgold [37] focuses on investigating the structure of the curriculum knowledge and the construction of curriculum in a tutoring system. They presents a system for tutoring a basic course in resistor network concepts. They have found four different views on their instruction: scientific laws, basic measurable properties, types of circuits, and types of the problems. Under these different views, the course is organized by the topics regarding to i) scientific laws; (e.g. Ohm's law, Kirchhoff's law, etc.); ii) current, voltage, and resistance, etc.; iii) the series and parallel circuits; and iv) the types of the problems presented to the student. (qualitative problems, quantity problems, etc..). They formulate a curriculum structure which has three layers of knowledge. The middle layer is the *curriculum goal lattice* which can incorporate a number of viewpoints on the goals of the instruction. An important feature

of this formulation is that the lowest level units, the simple lessons, are the same from the viewpoints. Like a Goal/Task Hierarchy, the connections between the goals are also created explicitly by the developers. However, the curriculum goal structure constructed is a goal lattice in which only the root can be associated with multiple views. In SQL–TUTOR, we have extended this feature by allowing multiple views to be associated with any non-unit topics.

9.2 Contributions

The first key point in this research is that we have provided a generic framework for KBTS construction which can be applied to different domains and showed how to implement them in a tutoring system. The system architecture is based on the view that a tutoring process consists of a series of communication cycles, and each cycle consists of four phases and focuses on one specific topic. We also clarified the roles played by each knowledge base and their interactions with the control procedures in Communication Controller.

Curriculum knowledge is an important part of knowledge for KBTSs which can help the system select appropriate topics to study and diagnose the student's mistakes. The Topic Association Graph (TAG) representation presented in this dissertation provides a framework for an instructor to explicitly encode his curriculum knowledge into the system's knowledge base. The TAG is an important part of knowledge for a KBTS and can help the system select appropriate topics to tutor and diagnose the student's mistakes. The instructor can incorporate his knowledge about the goal structure of a course, the multiple viewpoints on a topic, and the prerequisite relations among the topics into the curriculum knowledge base. Another feature of our formulation is that the precedence relation among topics are derived from the prerequisite relation over the domain concepts. The system will check the consistency property during the derivation process.

The current tools provided by SQL–TUTOR allow a student to select a specific topic and a specific view associated with the topic to study, or to eliminate a topic from his study. Therefore, the student can tailor the course curriculum based to his special background, requirements, and interests. In this way, the student can be more active and more involved in the tutoring, because he has a certain degree of freedom to select the path to accomplish the teaching goals associated with a course.

The student model in our system is built in the frame of the curriculum knowledge base. Compared with other systems in which the student models are created based on the domain knowledge bases, our approach has two prominent features:

1. the size of the student model has been reduced significantly, because the number of nodes in a *TAG* is much smaller than the number of items in a domain knowledge base (which is the number of the concepts and skills in the domain);

2. the S_1 and S_2 values of a node in a student model indicates the student's mastery on both a topic and its subtopics, whereas the traditional student models can only reflect the student's mastery on a particular item of the domain knowledge.

The domain knowledge is stored in the documents of the topics and the sample database in our system. The declarative knowledge is represented by the DESCRIPTION, SUPPLEMENT and **Notation** parts of the document. Especially, the relationships among the domain concepts are described by the CONTEXT parts of the documents, based on which the system can answer the various types of the questions from the student.

The procedural knowledge is stored in the EXAMPLE parts of the documents as the Semantic Query Graphs. A Semantic Query Graph can be used by the

system to derive the various SQL queries for a given problem, to evaluate the student performance, and to generate the meaningful feedbacks to the student.

9.3 Future Directions

The future directions of this research include:

1. to further test the feasibility of the system architecture proposed in this proposal. Although the current research focuses on tutoring SQL programming, we believe that the system architecture is generic and can be applied to many other domains. We are going to create tutoring systems for other domains after SQL-TUTOR has been fully developed;

2. to explore the feasibility of using neural networks [57] (or connectionist models [26]) to diagnose and correct the students' mistakes in problem solving. A neural network is based explicitly on an abstraction of current understanding of the information processing properties of neurons, which is a parallel, distributed information processing structure consisting of a large number of processing elements (PEs) interconnected together with unidirectional signal channels called connections. These PEs can have very high fan-ins and fan-outs and communicate with the rest of the network by transmitting a simple value. A PE transmits the same value to all PEs to which it is connected. All of the processing that goes on within each PE must be completely local, i.e., they must depend only upon the current values of the input signal and upon the values stored in the PE's local memory.

   Luc Steel points out in [65] that problem solving executed by connectionist network can be seen as a special example of heuristic classification which contains three problem solving steps:

- Abstraction of the data to classify the problem into a set of known categories (possibly one class);
- Association of the categories with a known (abstract) solution;
- Refinement of the solution to adapt it to the concrete case.

It seems quite favorable to apply neural network model approach for student evaluating and diagnosing because it fits into the heuristic classification very well:

- The set of solutions given by the student constitute the raw data to be abstracted, while the known categories correspond to the sets of student missing conceptions and misconceptions collected by the instructor;
- A remedying template can be created and associated with each category of missing conceptions and misconceptions and be retrieved based on the result of the data abstraction from the previous step;
- The remedying template can be instantiated by the data from the problem posed to the student, the student model, and student's response to form a concrete remedying plan. The execution of this plan is to correct the student's mistakes.

REFERENCES

1. J. R. Anderson, C. F. Boyle, and G. Yost, "The geometry tutor," in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California, pp. 1–7, 1985.

2. J. R. Anderson and E. Skwarecki, "The Automatated Tutoring of Introductory Computer Programming," *Communications of the ACM*, vol. 29, no. 9, pp. 842–849, 1986.

3. J. B. Black, D. S. Kay, and E. M. Soloway, "Goal and Plan Knowledge Representation: From Stories to Text Editors and Programs," *Interfacing Thought: Cognitive Aspects of Human-Computer Interactions* (J. M. Carroll ed.), The MIT Press, Cambridge, Massachusetts, pp. 36–60, 1987.

4. B. S. Bloom, "The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring," *Educational Researcher*, vol. 13, pp. 3–16, 1984.

5. B. J. Brecht, G. I. McCalla, J. E. Greer, and M. Jones, "Planning the Content of Instruction," *Artificial Intelligence and Education: Proceedings of the 4th International Conference on AI and Education*, IOS, Amsterdam, Netherlands; Springfield, Virginia, pp. 32–41, 1989.

6. J. S. Brown and K. VanLehn, "Repair Theory: A Generative Theory of Bugs in Procedural Skills," *Cognitive Science*, vol. 4, pp. 379–426, 1980.

7. J. Brown and R. Burton, "Diagnostic Models for Procedural Bugs in Basic Mathematical Skills," *Cognitive Science*, vol. 2, pp. 155–192, 1978.

8. J. Brown, R. Burton, and A. Bell, "SOPHIE: A Step Towards a Reactive Learning Environment," *International Journal of Man Machine Studies*, vol. 7, pp. 675–696, 1975.

9. J. Brown, R. Burton, and J. Kleer, "Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II and III," in *Intelligent Tutoring Systems* (D. Sleeman and J. S. Brown, eds.), Academic Press, London, 1982.

10. C. V. Bunderson, "The Design and Production of Learner-Controlled Courseware for the TICCIT System," *International Journal of Man-Machine Studies*, vol. 6, pp. 479–491, 1974.

11. M. L. Burger and J. F. Desoi, "The Cognitive Apprenticeship Analogue: a Strategy for Using ITS Technology for the Delivery of Instruction an as a Research Tool for the Study of Teaching and Learning," *International Journal of Man-Machine Studies*, vol. 36, pp. 775–795, 1992.

12. R. R. Burton, "Diagnosing Bugs in Simple Procedural Skills," in *Intelligent Tutoring Systems* (D. Sleeman and J. Brown, eds.), Academic Press, London, 1982.

13. R. R. Burton and J. S. Brown, "An Investigation of Computer Coaching for Informal Learning Activities," in *Intelligent Tutoring Systems* (D. Sleeman and J. Brown, eds.), Academic Press, London, 1982.

14. R. R. Burton and J. S. Brown, "A Tutoring and Student Modeling Paradigm for Game Environments," *Computer Science and Education* ACM SIGCSE *Bulletin*, vol. 8, no. 1, 1983.

15. J. J. Canas, M. T. Boja, and P. Gonzalvo, "Mental Models and Computer Programming," *International Journal of Human-Computer Studies*, vol. 40, pp. 795–811, 1994.

16. J. R. Carbonell, "AI in CAI: An Artificial Intelligence Approach to Computer Assisted Instruction," *IEEE Transactions on Man-Machine Systems*, vol. 11, pp. 190–202, 1970.

17. A. Cawsey, "The Structure of Tutorial Discourse," *Artificial Intelligence and Education: Proceedings of the 4th International Conference on AI and Education*, IOS, Amsterdam, Netherlands; Springfield, Virginia, pp. 47–53, 1989.

18. T. W. Chan, "Curriculum Tree: A Knowledge-Based Architecture for Intelligent Tutoring Systems", in *Intelligent Tutoring Systems: Second International Conference on Intelligent Tutoring Systems, ITS'92*, (C. Frasson, G. Gauthier and G.I. McCalla, eds.), Spring-Verlag, New York, pp. 140-147, 1992.

19. W. J. Clancey, "Tutoring Rules for Guiding a Case Method Dialogue," *International Journal of Man-Machine Studies*, vol. 11, pp. 25–49, 1979.

20. W. J. Clancey, "Tutoring Rules for Guiding a Case Method Dialogue," in *Intelligent Tutoring Systems* (D. Sleeman and J. Brown, eds.), Academic Press, London, 1982.

21. W. J. Clancey, *Knowledge-Based Tutoring: The GUIDON Program*, MIT Press, Cambridge, Massachusetts, 1983.

22. W. J. Clancey, J. J. Barnett, and P. R. Cohen, "Applications-Oriented AI Research: Education," in *The Handbook of Artificial Intelligence* (A. Barr and E. A. Feigenbaum, eds.), vol. 2, William Kaufmann Publishing Company, Los Altos, California, 1982.

23. A. Collins, "Processes in Acquiring Knowledge," in *Schooling and the Acquisition of Knowledge* (R. Anderson, R. Spiro, and W. Montage, eds.), Lawrence Erlnaum Associates, Hillsdale, New Jersey, 1979.

24. N. G. Craske, T. Richards, and G. Cumming, "ExperTutor, a Generic Purpose Intelligent Educational System", In *Advanced Research on Computer in Education* (R. Lewis and S. Otsuki, eds.), Elsevier Science Publishers, North-Holland, pp. 301–306, 1991.

25. W. J. Crowder, "Automatic Tutoring by Means of Intrinsic Programming," in *Automatic Teaching: The State of Art* (E. Galanter, ed.), Wiley, New York, 1959.

26. J. A. Feldman and D. H. Ballard, "Connectionist models and their implications: readings from cognitive science", in *Connectionism in Perspective* (D. Waltz and J. A, eds.), Ablex Publishing Corporation, Norwood, New Jersey, 1988.

27. K. Forbus and A. Steven, "Using Qualitative Simulation to Generate Explanations," Tech. Rep. 4480, Bolt, Beranek and Newman, Cambridge, Massachusetts, 1981.

28. D. Gentner and A. Stevens, *Mental Models*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1984.

29. I. P. Goldstein, "The Genetic Graph: A Representation for the Evolution of Procedural Knowledge," in *Intelligent Tutoring Systems* (D. H. Sleeman and J. Brown, eds.), Academic Press, London, 1982.

30. I. P. Goldstein and B. Carr, "The Computer as Coach: An Athletic Paradigm for Intellectual Education," in *Proceedings of the Annual Meeting the Association for Computing Machinery*, Seattle, Washington, 1977.

31. W. L. Johnson and E. Soloway, "PROUT: An Automatic Debugger for Pascal Programming," *Artificial Intelligence and Instruction: Applications and Methods*, Addison Wesley Publishing Company, Massachusetts, 1987.

32. D. M. Kaminski, "A Knowledge Base Approach to Learning to Program in Prolog," in *Computer Assisted Learning, 4th International Conference, ICCAL '92* (T. Tomek, ed.), Nova Scotia, Canada, Springer-Verlag, New York, 1992.

33. G. P. Kearsley, *Computer Based Training: A Guide to Selection and Implementation*, Addison Wesley Publishing Company, Massachusetts, 1983.

34. G. P. Kearsley, B. Hunter, and R. J. Seidel, "Two Decades of Computer Based Instruction Projects: What Have We Learned?" *T.H.E. Journal*, 1983.

35. J. A. Kulik, C. C. Kulik, and P. A. Cohen, "Effectiveness of Computer-based College Teaching: A Meta-analysis of Findings," *Review of Educational Research*, 1980.

36. J. Larkin, J. McDermott, D. P. Simon, and H. A. Simon, "Expert and Novice Performance in Solving Physics Problems," *Science*, pp. 1335–1342, June 1980.

37. A. M. Lesgold, "Toward a Theory of Curriculum for Use in Designing Intelligent Instructional Systems", in *Learning Issues for Intelligent Tutoring Systems*, (H. Mandl and A. M. Lesgold, eds.), Springer-Verlag, New York, 1988.

38. D. C. Littman, J. Pinto, and E. Soloway, "An Analysis of Tutorial Reasoning About Programming Bugs," in *Proceedings of the Eighth National Conference on Artificial Intelligence*, Philadelphia, Pennsylvania, 1986.

39. P. Marcenac, "An Authoring System for ITS Which is Based on a Generic Level of Tutoring Strategies," in *Computer Assisted Learning, 4th International Conference, ICCAL '92* (T. Tomek, ed.), Nova Scotia, Canada, Springer-Verlag, New York, 1992.

40. K. V. Marcke, "Instructional Intelligent Expertise," *Tutoring Systems: Second International Conference on Intelligent Tutoring Systems, ITS'92*, Spring-Verlag, New York, 1992.

41. J. Mcdonald, "The EXCHANGE CAI System," in *University-Level Computer-Assistance Instruction at Stanford: 1968-1980* (P. Suppes, ed.), Institute for Mathematical Studies in the Social Sciences, Stanford University, Stanford, California, 1981.

42. M. D. Merrill, E. W. Schneider, and K. Fletcher, *TICCIT*, Englewood Cliffs, New Jersey, 1980.

43. R. Moyse, *Knowledge Negotiation Implies Multiple Viewpoints*, IOS, Amsterdam, Netherlands, Springfield, Virginia: IOS, Amsterdam, Netherlands, pp. 140–149, 1989.

44. R. Moyse and M. Elson-Cook, "Knowledge Negotiation: An Introduction," *Knowledge Negotiation*, (R. Moyse and M. Elson-Cook eds.), Academic Press, New York, pp. 1–19, 1992.

45. W. R. Murray, "Control for Intelligent Tutoring Systems: A Blackboard-based Dynamic Instructional Planner," in *Artificial Intelligence and Education: Proceedings of the 4th International Conference on AI and Education* (D. Bierman, J. Breuker, and J. Sandberg, eds.), IOS, Amsterdam, Netherlands, May 1989.

46. W. R. Murray, "A Blackboard-based Dynamic Instructional Planner," in *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Boston, Massachusetts, 1990.

47. H. S. Nwana, "FITS: A Fraction Intelligent Tutoring System," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1991.

48. J. Orlansky and J. String, "Computer-based Instruction for Military Training," *Defense Management Journal*, 1981.

49. O. C. Park, R. S. Perez, and R. J. Seidel, "Intelligent CAI: Old Wine in New Bottles, or a New Vintage," *Artificial Intelligence and Instruction: Application and Methods*, Addison-Wesley Publishing Company, Massachusetts, 1987.

50. D. R. Peachey and G. I. McCalla, "Using Planning Techniques in Intelligent Tutoring Systems," *International Journal of Man–Machine Studies*, vol. 24, pp. 77–98, 1986.

51. P. G. Polson, "A Quantitative Theory of Human-Computer Interaction," *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, The MIT Press, Cambridge, Massachusetts, 1987.

52. P. J. Pratt, *A Guide to SQL*, Boyd and Fraser Publishing Company, Boston, Massachusetts, 1989.

53. B. J. Reiser, J. R. Anderson, and R. G. Farrell, "Dynamic Student Modeling in an Intelligent Tutor for LISP Programming," in *Proceeding of the Eighth International Joint Conference on Artificial Intelligence*, Los Angeles, California, 1985.

54. C. B. Schwind, "An Intelligent Language Tutoring System," *International Journal of Man-Machine Studies*, vol. 33, pp. 557–579, 1990.

55. R. J. Seidel, *Current Status of Computer-Administered Instruction Work Under Project IMPACT*, Human Resources Research Organization, Alexandria, Virginia, 1971.

56. V. J. Shute, "Regarding the I in ITS: Student Modeling," in *Proceeding of ED-MEDIA 94 - World Conference on Educational Multimedia and Hypermedia*, Vancouver, British Columbia, Canada, 1994.

57. P, K. Simpson, *Artificial Neural Systems*, Pergamon Press, New York, 1990.

58. D. H. Sleeman, "Intelligent Tutoring Systems: A Review," in *Proceedings of the EdCompCon '83 Meeting*, 1983.

59. D. H. Sleeman and J. S. Brown, *Intelligent Tutoring Systems*, Academic Press, London, 1982.

60. D. H. Sleeman and R. J. Hendley, "ACE: A System Which Analyses Complex Explanations," in *Intelligent Tutoring Systems* (D. H. Sleeman and J. Brown, eds.), Academic Press, London, 1982.

61. E. Soloway, "Learning To Program = Learning To Construct Mechanisms and Explanations," *Communication of the ACM*, vol. 29, pp. 850–858, September 1986.

62. E. Soloway, K. Ehrlich, J. Bonar, and J. Greenpan, "What do Novice Know About Programming," *Directions in Human/Computer Interaction* (A. Badre and B. Shneiderman eds.), Ablex Publishing Corporation, Norwood, New Jersey, pp. 27–54, 1982.

63. J. C. Spohere and E. Soloway, "Simulating Student Programmers," in *Proceeding of the Ninth International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989.

64. J. C. Stansfield and I. Goldstein, "Wumpus Advisor I: a First Implementation of a Program That Tutors Logical and Probabilistic Reasoning Skills," in *AI Lab Memo 381*, MIT Press, Cambridge, Massachusetts, 1979.

65. L. Steel, "Connectionist Problem Solving – An AI Perspective", in *Connectionism in Perspective* (D. Waltz and J. A, eds.), Ablex Publishing Corporation, Norwood, New Jersey, 1988.

66. J. Stelzer and J. Garneau, "Project IMPACT Software Documentation: Overview of the Computer Administered Instruction Subsystem," Tech. Rep. 72–21, Human Resources Research Organization, Alexandria, Virginia, 1972.

67. A. L. Stevens, A. Collins, and S. Goldin, "Misconceptions in Student's understanding," *International Journal of Man-Machine Studies*, vol. 11, pp. 145–156, 1979.

68. A. L. Stevens and A. Collins, "The Goal Structure of a Socratic Tutor," in *Proceedings of the Association for Computing Machinery Annual Conference*, 1982.

69. N. A. Streitz, "Mental models and metaphors: Implications for the design of adaptive user-system interface," in *Learning Issues for Intelligent Tutoring Systems* (L. Mandl, ed.), Springer-Verlag, New York, 1988.

70. P. Suppes, M. Jerman, and D. Brian, *Computer-assisted Instruction: The 1965-66 Stanford Arithmetic Program*, Academic Press, New York, 1968.

71. P. Suppes and M. Morningstar, *Computer-assisted Instruction at Stanford: 1966-68: Data, Models and Evaluation of Arithmetic Program*, Academic Press, New York, 1972.

72. T. Tokuda and A. Fukuda, "A Probabilistic Inference Scheme for Hierarchical Buggy Models," *International Journal of Man-Machine Studies*, vol. 38, pp. 857–872, 1993.

73. R. F. van der Lans, *Introduction to SQL*, Addison-Wesley Publishing Company, Massachusetts, 1988.

74. Y. Visetti and P. Dague, "Plan Inference and Student Modeling in ICAI," in *Proceedings of Ninth National Conference on Artificial Intelligence*, Seattle, Washington, 1987.

75. E. Wenger, *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*, Morgan Kaufmann Publishers, Los Altos, California, 1987.

76. B. Y. White and J. R. Frederiksen, "Modeling Expertise in Troubleshooting and Reasoning about Simple Electric Circuit," in *Proceedings of the Sixth Cognitive Science Conference*, Boulder, Colorado, pp. 337–343, 1984.

77. B. Y. White and J. R. Frederiksen, "Intelligent Tutoring Systems Based upon Qualitative Model Evaluations," in *Proceedings of the Eighth National Conference on Artificial Intelligence*, Philadelphia, Pennsylvania, 1986.

78. B. Woolf, "Theoretical Frontiers in Building a Machine Tutor," *Artificial Intelligence and Instruction: Application and Methods*, Addison-Wesley Publishing Company, Massachusetts, 1987.

79. B. Woolf, D. Blegen, J. Jansen, and A. Verloop, "Teaching a Complex Industrial Process," in *Proceedings of the Eighth National Conference on Artificial Intelligence*, Philadelphia, Pennsylvania, 1986.

80. Y. Yano, A. Kashhara, and W. McMichael, "Stabilizing Student Knowledge in Open Structured CAI," *International Journal of Man-Machine Studies*, vol. 37, pp. 595–612, 1992.

81. G. Zhou, "Curriculum Knowledge Representation in SQL–TUTOR," in *Proceeding of ED-MEDIA 94–World Conference on Educational Multimedia and Hypermedia*, Vancouver, British Columbia, Canada, 1994.

82. G. Zhou, J. T. L. Wang, and P. A. Ng, "A Knowledge-Based Tutoring System for SQL Programming," in *Proceedings of 6th IEEE International Conference on Tools with Artificial Intelligence*, New Orleans, Louisiana, 1994.

83. G. Zhou, J. T. L. Wang, and P. A. Ng, "Curriculum Knowledge Representation and Manipulation in Knowledge-Based Tutoring System," to appear in *IEEE Transactions on Data and Knowledge Engineering*, 1996.