Interactive Correction and Recommendation for Computer Language Learning and Training

Claus Pahl, *Member, IEEE*, and Claire Kenny

Abstract—Active learning and training is a particularly effective form of education. In various domains, skills are equally important to knowledge. We present an automated learning and skills training system for a database programming environment that promotes procedural knowledge acquisition and skills training. The system provides meaningful, knowledge-level feedback such as correction of student solutions and personalised guidance through recommendations. Specifically, we address automated synchronous feedback and recommendations based on personalised performance assessment. At the core of the tutoring system is a pattern-based error classification and correction component that analyses student input in order to provide immediate feedback and in order to diagnose student weaknesses and suggest further study material. A syntax-driven approach based on grammars and syntax trees provides the solution for a semantic analysis technique. Syntax tree abstractions and comparison techniques based on equivalence rules and pattern matching are specific approaches.

Index Terms— Artificial Intelligence - Applications and Expert Knowledge-Intensive Systems [I.2.1], Data Structures [E.1], Education [J.1.b], Programming languages [D.3], Query languages [H.2.3.e].

1 Introduction

THE delivery of feedback is an integral part of learning processes. Feedback should be relevant, precise and understandable. The level a student reaches when learning is often proportional to the student's engagement with a teacher or an activity. In computer-aided learning and training, feedback is of central importance in particular if a human tutor is not always available [15].

We present an automated, computer-based tutoring system that supports a skills training environment for the database language SQL.

- In particular, it provides feedback for the student that is meangingful and of a contextually high quality. The system allows a knowledge- or skills-level interaction with the content through programming activity and synchronous contextual feedback [27].
- An automated tutoring process allows students to individually tailor their learning environment by defining feedback preferences and choosing their own learning paths through the course curriculum.

The student benefits from a system that is always available and that analyses and corrects a submission and offers feedback and personalised guidance and recommendations based on the analysis results.

Formally defined languages are particularly suitable to be supported by automated tutoring systems. Computer languages such as many specification, modeling and programming languages fall into this category. SQL is in this context a language of medium complexity. SQL is a suitable topic to explore these issues, but they apply equally to other computer-processable languages, ranging from textual to graphical languages [17].

Our primary objective is to investigate an integrated approach to correction, domain-specific feedback and personalised guidance features. At the core of this approach is a correction technique that allows personalised domain-specific feedback and guidance.

- We develop techniques to analyse the SQL select statement in order to identify problems that a typical student might encounter while trying to solve SQL programming problems. These errors are categorised according to a multi-dimensional error classification scheme.
- We determine adaptivity techniques for use in a knowledge-based feedback system for correction and recommendations.

We introduce the underlying data structures and analysis techniques for correction and personalized recommendation. A pattern-based error classification and correction component analyses student input in order to provide immediate feedback. This technique can also be used to diagnose student weaknesses and recommend further study material. A syntax-driven approach based on grammars and syntax trees provides the solution for a semantic analysis technique. Syntax tree abstractions are the central data structures that represent student answers (in terms of SQL) to a given set of problems. Two central comparison, correction, and diagnosis techniques are introduced:

- equivalence rules on syntax trees to determine semantical equivalence of solutions,

- *C. Pahl is with Dublin City University, Dublin 9, Ireland. E-mail: cpahl@computing.dcu.ie.*
- *C. Kenny is with Dublin City University, Dublin 9, Ireland. E-mail: ckenny@computing.dcu.ie.*

Manuscript received (insert date of submission if desired).

- pattern matching to localize and classify errors.

In our presentation, we focus on data representation and data processing aspects. Architectural issues are only sketched.

We start by introducing the pedagogical framework in Section 2. Section 3 outlines the information and system architecture. In Section 4, we present our correction solution as a local, immediate form of feedback. In Section 5, we then address recommendation as global, summative form of feedback. In Section 6, we discuss potential and weaknesses and also a range of related systems, before ending with some conclusions.

2 Framework

The application that provides the context of our investigation is an automated SQL tutor, which is part of the Interactive Database Learning Environment IDLE. We present an IDLE overview and its pedagogical principles in this Section.

2.1 SQL Learning and Training

IDLE is the Interactive Database Learning Environment, an online support system for a database course that is in use since 1999 [25]. Database programming and querying is a core skill for computer scientists and engineers.

Computer-supported formal computer language learning and training is the IDLE objective. A central success factor is knowledge-level interaction, i.e. interactions between student and system in terms of concepts and objects that have a meaning in the subject domain [26]. In this case, database objects and SQL language expressions and statements are at the core of the student-system communication.

An intelligent tutoring system supports the SQL programming features through online exercises [18]:

- It provides a range of SQL programming problems, each addressing specific language constructs.
- It corrects student answers (which are submitted electronically through a Web-based system) and executes them (using an attached database server).
- It gives recommendations at the end of each lesson (consisting of a range of suitable problems based on identified weaknesses).

A screenshot of a submission that gives a limited level of feedback is shown in Fig. 1 (feedback level are determined based on user preference and a pedagogical strategy).

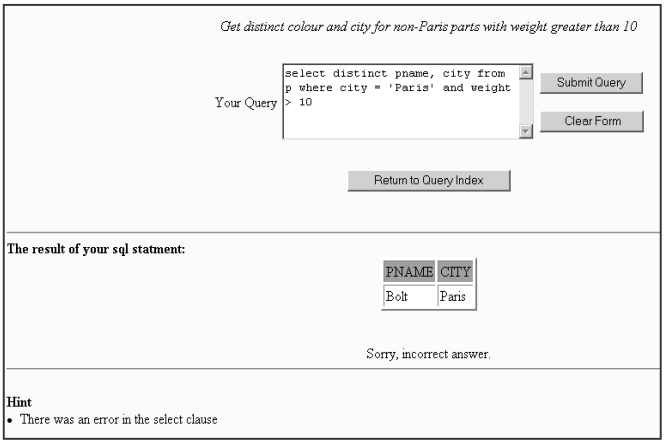

Fig. 1. IDLE - SQL Tutor (Screenshot)

2.2 Apprenticeships and Scaffolding

Stephenson [31] argues that experience is the foundation of and the stimulus for learning. Learning is primarily developed through activity.

Skills training involves higher levels of activity and interaction than typical acquisition of declarative knowledge. A student trains by practising a task. An apprenticeship as a type of student is concerned with procedural knowledge acquisition and skills training. Traditional apprenticeship is a form of teaching and learning that has been used successfully throughout the ages, primarily for practical tasks. Apprenticeship is a three-step process involving a master and an apprentice. Initially, the master demonstrates the completion of the stages of a task while the apprentice observes. Then, apprentice works at the task while the master observes and offers advice. The apprentice practises in a controlled environment. Finally, the apprentice eventually achieves competency and self-reliance.

Apprenticeships can be realized as a blend of scaffolding, fading and coaching.

- Scaffolding is a temporary support while completing a task or activity. The key idea behind scaffolding is to provide a student with timely support at an appropriate level. Collins et al. [12] refer to scaffolding as being a set of limited hints and feedback.
- Ideally, scaffolding will be faded, meaning it will be removed gradually, thus encouraging the student to work in a self-reliant manner.
- Coaching is the process of overseeing the student's learning [12], [13]. It involves formulating the course of work the learner should take, providing timely scaffolding at the appropriate level and fading it accordingly, and offering meaningful feedback and encouragement.

Cognitive apprenticeship moves the traditional apprenticeship into the classroom and the cognitive domain [12]. A major principle of cognitive apprenticeship is collaboration and conversation with a master [33]. Cognitive apprenticeship uses the idea of situated learning, whereby the learner is placed in a real-world environment [13].

The virtual apprenticeship model [23] applies cognitive apprenticeship to the Web context, and is therefore a suitable concept for Web-based learning. This model uses scaffolding and activity-based learning to allow the student to construct knowledge, practise skills and gain experience in an online environment. The construction of artefacts (such as software, but also other digital artifacts or representations of concrete objects) and a realistic or even authentic setting are vital [16].

2.3 Intelligent Tutoring and Pattern Matching

An Intelligent Tutoring System (ITS) is a computer-based

instructional system with models of instructional content that specify what to teach, along with teaching strategies that specify how to teach [22]. ITSs have been shown to be highly effective [5], although ITS in the past have often been restrictive, limiting the student's control of the learning experience [8]. A traditional ITS has four distinct components [6]:

- The expert model, or domain model, contains knowledge of the domain or subject area [11].
- The student model holds information about the student (personal details, learning preferences), along with a representation of the knowledge s/he holds.
- The pedagogical model determines when and how to instruct the student. It makes decisions about the topic, the problem, and feedback.
- The interface acts as the means of communication between the student and the ITS.

Pattern matching or pattern recognition [22] is a method that can be used in ITS to define ideal solutions or ideal learning paths. For instance, it can be used as a means of correcting student work. CAPIT's student modeller [19], for instance, is a pattern matcher that takes a student solution to a problem and determines which constraints are violated. Pattern matching can also be used to ascertain a higher level of student understanding. The Tactical Action Officer TAO [32] applies pattern-matching rules to detect sequences of actions that indicate whether the student understands an activity.

3 Information and System Architecture

The information and system architectures provide the foundations for the feedback techniques.

3.1 Information Architecture

We propose a language-driven approach to e-learning. Languages and their representations in terms of grammars and syntax trees provide the data and knowledge structures of the approach. Correction and recommendation techniques provide feedback for the student based on these data structures that capture the formal aspects of content knowledge.

A two-layered model is at the core of the approach:

- the content (or domain) model captures language expressions, which are defined by a grammar, in terms of syntax trees and their abstractions to enable analyses and personalised feedback,
- the student model is a meta-model capturing feedback preferences chosen by the student and observed student performance within the exercises given by the system.

The representation of content knowledge as structured data in the form of syntax trees is the crucial aspect that enables knowledge-level interactions between student and system and intelligent feedback in the form of correction and recommendation.

3.2 System Architecture

IDLE is a larger system of which the SQL tutor is only one component. The SQL tutor itself is a componentized system comprising of:

- an interface that provides the student with a lesson consisting of a range of individual SQL programming problems,
- a student component that manages the student model, i.e. which keeps preferences up-to-date and which tracks the student performance in the system,
- a correction component that receives a student answer for a particular problem and that semantically analyses the answer in order to provide meaningful feedback,
- a guidance and recommendation component which, based on an assessment of student weaknesses provides recommendations for further study for the student.

The architecture will be illustrated in more detail in the two central technical Sections 4 and 5 (see Figs. 2 and 5, respectively).

4 Content and Language

A computer language is at the core of our learning scenarios, i.e. formal language representations in terms of grammars and tree-based structures form the core content data and knowledge structure used in facilitating the language learning and training experience.

We present basic language representation principles before introducing the database language SQL as a learning and training subject. We discuss learning problems and student answers and provide a syntax-driven technique to correct answers based on ideal solutions using a pattern-matching approach.

4.1 Classification and Correction Architecture

While we do not investigate the tutoring system from a software architecture perspective, we look at the architecture first to outline and structure information and processing components and their interactions to set the scene for the data and knowledge aspects of feedback generation.

The correction architecture is presented in Fig. 2. In terms of standard ITS components from Section 2.3, the student model and the interface are directly represented, the expert model is spread over problem repository and ideal solution repository and the rules embedded in the correction component. The pedagogical model is implemented partly by the correction component (which localized and categorized errors) and partly by the recommendation component (which we discuss in Section 5).

4.2 Language Representation

The computer languages we consider here as the subject of learning and training are computer-processable, formally defined languages for the specification, modeling or implementation of computer systems. We provide some general background here.

We can distinguish three facets of computer languages here [2]:

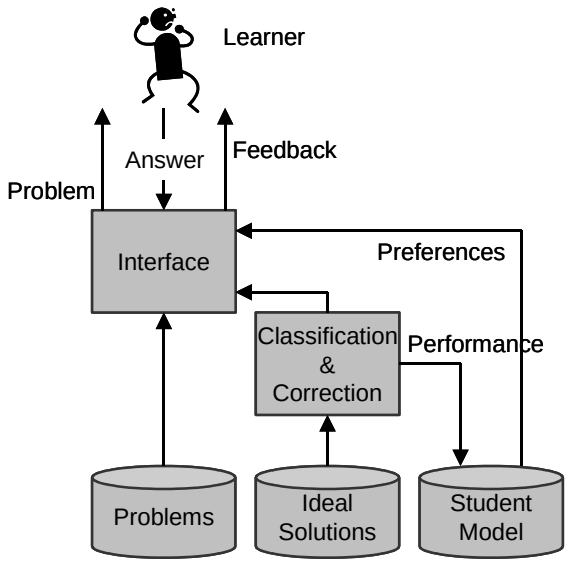

Fig. 2. Correction Architecture

- Syntax refers to the lexical and structural aspects. Based on a vocabulary of keywords and user-defined elements, sentences in the language can be formulated. The construction of these sentences is governed by a grammar.
- Semantics refers to the meaning of elements and sentences of a language. This is often done mathematically or by mapping onto other notations.
- Pragmatics refers to the use of the language. It captures general rules and guidelines that, for instance, improve the readability of a sentence or the quality on an expression.

Abstract and concrete syntax are distinguished. Concrete syntax is concerned with the actual representation as it is provided by the user, i.e. with keywords and other lexical concerns such as the construction of identifiers. Concrete syntax validation is usually supported by language processors such as syntax checkers, compilers or execution tools [2]. We ignore this aspect. We are concerned here with abstract syntax, which can be defined in terms of grammars, i.e. production rules that, if applied, lead to grammatically correct sentences. Abstract syntax trees are representations of grammatically correct sentences.

Abstract syntax trees (ASTs) as representations of language expressions shift the focus from lexical concerns to structural notational ones [2], [3]. In some cases, further abstractions of an AST can take place to facilitate the application of specific analyses or transformations. For instance, grouping of tree elements and their classification can take place – an aspect that we use later on to remove irrelevant detail for the correction technique.

4.3 Language – Problems and Error Classification

Our focus is language semantics and its comprehension. Although syntax and pragmatics are important, we provide a semantics-specific solution to error classification. SQL-specific language aspects, based on Section 4.2, shall be introduced here.

```
query_spec
        : SELECT selection table_exp

selection
        : column_ref_commalist

table_exp
        : from_clause opt_where_clause

from_clause
        : FROM table_ref_commalist

where_clause
        : WHERE search_condition
```

Fig. 3. Simplified SQL Grammar (Excerpt)

The select statement is a fundamental SQL statement, used to query and extract information from a database [1], [26]. It can be made up of six clauses, but we focus here on the three central ones – SELECT, FROM, WHERE – whereas the others are dealt with in [17]. A simplified grammar of a few higher-level production rules is presented in Fig. 3. These three clauses map to input elements (FROM), a condition that is the extraction filter (WHERE), and an output description (SELECT). An example is:

```
SELECT    colour
FROM      parts
WHERE     weight > 10 and city = "Paris".
```

A list of database table names can be provided in the FROM clause. The SELECT clause can contain a list of column names of tables named in the from-list, possibly combined with aggregation operators such as minimum or average. The qualification in the WHERE clause is a condition or filter constraint based on logical and comparison operators.

We distinguish here two different notational elements in SQL that are of relevance for the error classification in the correction technique:

- Elements – objects and functions. SQL refers to a number of different data objects, mainly based on the Relational Algebra as SQL's mathematical foundation. This includes objects such as tables and attributes. We also add functional elements to this category, such as aggregate functions or comparison operators. These elements are part of the database object platform.
- Clauses. SQL has a fixed syntactical structure based on the SELECT … FROM … WHERE template with in total six clauses. Each of the clauses focuses on a specific step in query processing. The FROM clause defined the input tables, SELECT constructs the output and WHERE defines filtering and selection conditions. These elements are part of computational query processing.

These dimensions are motivated by the experience and

observations of educators based on their experience in teaching SQL and the common difficulties that they have encountered. This classification helps to categorise and localize common learner errors and problems by distinguishing object and computational aspects.

Our tutor is a problem-based learning and training environment. A number of data query problems are provided in a repository; each problem typically addresses a specific SQL language aspect. An entry in the problem repository contains the following information:

- the problem itself, formulated as a natural language sentence, which is going to be made available to students, see Fig. 1,
- problem metadata including a problem focus, which is expressed in terms of SQL elements and clauses (this characterization reflects the educator's defined learning goal for a problem).

An example for the language aspect categorisation shall illustrate the two aspect dimensions. Consider a table s that captures information about suppliers of parts.

- Question: Get the numbers and names of all suppliers. An (incorrect) student answer could be:

  SELECT sno, name FROM s

  The diagnosis would recognize this statement as incorrect because the student has tried to select the attribute "name". We assume that the correct attribute is "sname". This can be identified as primarily a function-specific error in the select clause, i.e. the output of the statement is affected through an attribute element.
- Question: Get the maximum status of suppliers in Paris. An (incorrect) student answer could be:

  SELECT min(status) FROM s WHERE city = 'Paris'

  In this case, the student has selected the minimum status instead of the maximum status. Here, the system identifies a primarily element-specific semantic error due to the misuse of the aggregate function "min" on element "status" in the select clause.

The language aspect dimensions are the core mechanism to locate and categorize student errors. The identification of the primary error is based on heuristics defined by database educators. The different categories of errors should be identified by the tutor when a student has made more that one mistake. The number and ordering of displayed errors depends on the student's preferences and a heuristics-based prioritisation of errors by the instructor.

4.4 Language – Solutions and Errors

The previous two examples have clarified the requirements for a correction approach. We propose the following set-up based on problems and solutions:

- Problems: an empirical problem determination and definition is based on an educator's judgement and experience [28].

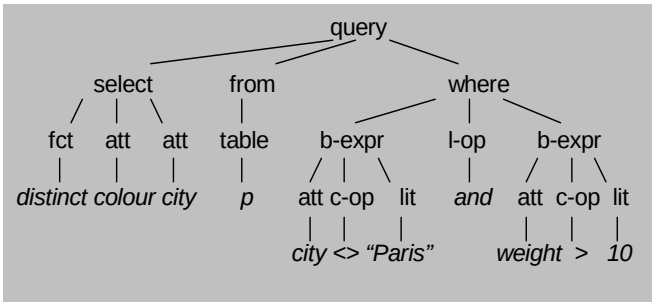

Fig. 4. Simplified Abstract Syntax Tree (SAST)

- Solutions: for each problem, the educator defines an ideal solution.

Our focus is on semantics and the correction of semantic errors in SQL statements. Database management systems usually provide feedback on syntax. Pragmatic aspects of language use require the student's comprehension of syntax and semantics, and shall therefore be neglected in this investigation.

We propose to use the ideal solutions to identify student errors. This general idea stems from intelligent tutoring systems. In its application of fomalised languages, two techniques are required:

- a notion of semantic equivalence on answer representations (syntax trees) to identify correct answers that do not syntactically equal to the ideal solution
- a matching technique on syntax trees that compares student answer and ideal solution by matching structural patterns in the two syntax tree representations to identify correct and erroneous elements.

An error categorization scheme, based on elements and clauses, allows classifying and explaining errors. Since multiple errors are always possible, a weighted categorization based empirically on the educator's experience is needed.

4.5 Pattern-based Correction

At the core of our approach is a syntax-driven analysis technique to identify and correct semantic student errors. Although semantics is the issue, we require only an ideal solution in syntactical representation from the educator. A comparison of ideal solution and student answer on a syntax level is the central correction activity for semantic error [2].

- The ideal solution (syntactic representation) acts as a semantic solution for the problem where one syntactic example denotes the semantics.
- Semantics-based equivalence and matching rules enhance the ideal solution and allow the semantic error detection and correction to be carried out.

The principle idea is to match student answer and ideal solution using syntactical patterns in the comparison of syntax trees of both query expressions.

4.5.1 Answer and Solution Representation

The data representation structures that capture content

knowledge are based on:

- Language grammars to define the notation through which learner and system interact. Here the BNF grammar for ISO/IEC 9075 Database Language SQL standard forms the basis [1], which we have simplified in our prototype implementation.
- Abstract syntax trees (ASTs) are tree-based representations of concrete sentences in a language defined through the grammar production rules [2], [3]. ASTs abstract from concrete syntax aspects such as keywords.
- Simplified abstract syntax trees (SASTs) are a form of syntax trees that we introduce here to provide a further level of abstraction. For instance, literals of different types are not distinguished for our correction, but which are typically distinguished in ASTs for code generation. Here, we introduce a generalized class Literal for string and numerical literals. SAST filter out information that is not required for the correctness analysis.

We define the syntax trees AST and SAST and also the AST and SAST construction process as follows:

- a syntax tree is defined as an acyclic graph $ST = (N,E,R)$ with nodes N and edges $E \subseteq N \times N$ where one vertex $R \in N$ singled out as the root of the tree,
- root R = 'query' to represent an SQL query according to the SQL grammar,
- the $AST = (N_{AST},E_{AST},R)$ is defined as usual in compiler construction, e.g. by removing all keyword nodes (e.g. SELECT, FROM, WHERE in Fig. 2) from N and all edges from E that include these nodes,
- the $SAST = (N_{SAST},E_{SAST},R)$ is defined by merging nodes into equivalence classes (e.g. string_literal and numerical_literal are merged into literal); such nodes from N_{AST} are replaced by their equivalence classes to form N_{SAST}; accordingly, all edges from E_{AST} that include these nodes are replaced by equivalence classes to form E_{SAST}.

4.5.2 Answer Processing ad Pattern Matching

In the SQL tutor, answers to problems are processed as follows. Student answers are submitted to the tutoring system via the interface and are processed by the correction component. A syntactically correct query (this is ascertained by an associated database server that is utilized to carry out the syntax check) is matched against an ideal solution for possible semantic errors.

- A parser in the correction component splits the query into segments based on the clauses. The parser also partitions the ideal solution from the ideal solution repository.
- Having been split into segments, the student answer and the ideal solution are normalised for easier matching. This step consists of removing an extraneous elements and standardising certain miscellaneous items and operators. Some simple equivalence rules (see below in Section 4.5.3) are already applied at this stage to minimize the complexity at later processing stages.

Our correction approach uses a pattern matching technique. Abstracting student answers and ideal solutions in two steps – abstracting from concrete syntax (AST) and from irrelevant information (SAST) – provides the basis of the comparison. A pattern matching on the SAST structure acts as a similarity measure between ideal solution and student answer. A two-step procedure is applied:

1. semantic equality determination (Section 4.5.3),
2. semantic inequality determination (Section 4.5.4).

Both error determination (semantically correct or incorrect) in Step 1 and error localization and categorisation in Step 2 are dealt with in order to provide immediate, meaningful feedback to the student.

4.5.3 Semantic Equality Determination

A number of equivalence rules define semantical equality as SQL statements can usually be syntactically modified without changing their semantics. The rules define an equivalence relation $\approx \subseteq SAST \times SAST$. These rules include:

- ORD-1: reordering of attributes in the SELECT clause
- ORD-2: reordering of table names in the FROM clause
- ORD-3: reordering of Boolean expressions in the WHERE clause
- EQU-1: equivalence of comparison operators in the WHERE clause
- OPT-1: optional use of prefixes/aliases in all clauses

These are some of the most common rules. These are formulated formally as indicated by the following example for the first rule ORD-1

$$\begin{aligned} &((select, att1, att2) , ([select,att1] ; [select,att2])) \\ \approx \\ &((select, att1, att2) , ([select,att2] ; [select,att1])) \end{aligned}$$

for nodes *select*, *att1* and *att2* from N_{SAST} and corresponding edges in E_{SAST} based on the nodes . We use a semicolon to express ordering of subtrees.

For the last rule OPT-1, a conditional equivalence rule is defined as follows

$$((table\_ref), ()) \approx ((prefixed\_table\_ref)$$

if $table\_ref.ID = prefixed\_table\_ref.ID$ and $unique(table\_ref.ID)$

where

- *table_ref* and *prefixed_table_ref* are suitable nodes
- the postfix *ID* is an attribute of the node that denotes the concrete table identifier
- *unique* is a predicate that flags if the identifier is uniquely applied in the sentence under consideration

The sample, formalized rules are rather simple rules. However, more complex semantic equivalences exist between SQL queries. A typical example is the equivalence of a nested and a non-nested query. For instance, the following

SELECT colour FROM p WHERE pno in (
SELECT pno FROM sp)

≈

SELECT colour FROM p,sp WHERE p.pno = sp.pno

are equivalent queries. This can also be captured by an equivalence rules, although a more complex one:

(*N1* = (*table_ref_list1*, *nested_in*(*c*), *table_exp*(*c*), *table_ref_list2*) , *E1* = (…))

≈

(*N2* = (*table_ref_list3*, *equality*(*t1.c*, *t2.c*)) , *E2* = (…))

if

$table\_ref\_list1 \subseteq table\_ref\_list3$ and
$table\_ref\_list2 \subseteq table\_ref\_list3$ and
$table\_ref\_list1 \cap table\_ref\_list2 = table\_ref\_list3$

where

- *c* is a column name
- *nested_in*(*c*) is a macro-style abbreviation that refers to the SQL IN-operator applied to column *c*
- *equality*(*t1.c*, *t2.c*) is a macro-style abbreviation that refers to the equality comparison operator for tables $t1, t2 \in table\_ref\_list3$
- *table_ref* and *table_exp* are expressions according to the SQL grammar

As we can see, the two SASTs (*N1,E1,R1*) and (*N2,E2,R2*) are incompletely specified and some macro abbreviations have been included to shorten the construction. Rules can get complex as a consequence of the language definition. This example, however, illustrates the idea of pattern matching. The macros *nested_in* and *equality*, for instance, stand for two patterns that semantically correspond to each other. For this presentation, we have made these patterns explicit by providing names for them.

There is a range of other equivalences, essentially based on the availability of additional operator that simplify expressions or accommodate particular styles. The JOIN … USING construct for the FROM clause is an example.

We cannot present all rules here that define semantic equivalences for SQL queries. The quantity and the complexity (of some) of the rules create an accuracy problem from the implementation perspective, which we discuss in Section 6.

The application of these rules on a comparison of two SASTs allows semantically correct student answers that are syntactically different from the ideal solution to be identified. The rule application follows the following strategy:

- the ideal solution SAST is the starting point of the comparison
- rules are applied bottom-up to the SASTs, e.g. the comparison operator equivalence is applied earlier (at a lower level) than reordering of Boolean expressions
- most rules are specific to SQL clauses, e.g. to 'select', 'from' or 'where', which means that rules are applied locally
- a simple rule preferences list defines the remaining rule application ordering

We define as *IdealSolution** the equivalence class $IdealSolution^* = \{ SAST \mid SAST \approx SAST_{IDEAL} \}$. All answers represented as a $SAST \in IdealSolution^*$ are considered correct. The previous rule application strategy determines semantic correctness.

4.5.4 Semantic Inequality Determination

If the application of all equivalence rules does not solve the correction problem, then a semantic error can be assumed for the student answer. A pattern matching approach shall serve to

- locate the error(s) within the SAST: if a subtree of the answer SAST cannot be matched with the corresponding ideal solution SAST based on the equivalence rule, then an error has been located and the corresponding position in the student answer can be identified.

  A node $err \in N$ for $SAST = (N,E,R)$ contains an error, if

  $$\exists\, n \in SAST_{IDEAL} \,.\, Context(err) = Context(n)$$

  and

  $$err \in Nodes(SAST_{IDEAL})$$

  where
  - *Context* represents the parents of a node up to the root,
  - *Nodes* projects onto the nodes of a tree.
- classify the error(s) within the syntactical structure: the abstraction and expression classification in the SAST allows the association of an error to the relevant semantic construct – which is the parent node in the SAST.

  An error occurring in a construct is determined as follows using function *ErrorType*:

  $$ErrorType(err) = \begin{cases} Parent(err) & \text{if } Parent(err) \in Nodes(SAST_{IDEAL}) \\ ErrorType(Parent(err)) & \text{otherwise} \end{cases}$$

  where
  - *Parent* represents the parent of a node.

Since multiple errors are possible, an error weighting and ranking technique is required. Error categories and associated weights (based on error categories for structure and object dimensions) are determined (usually empirically) based on the educator's experience and can be individually configured. We distinguish overall two forms of feedback:

- correction: immediate feedback that can comprise error location, error explanation, hints and par-

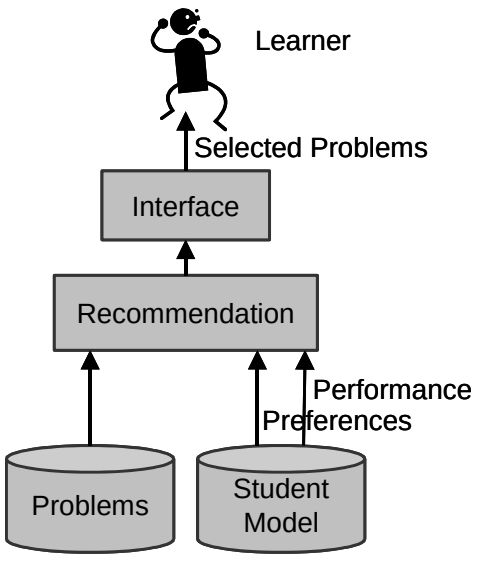

Fig. 5. Recommendation Architecture

tial/full correct solutions,
- recommendation: accumulative feedback that can comprise a diagnosis of past performance and recommendations for future studies (in the form of selected exercises).

5 Learner and Interaction

The correction technique provides immediate, synchronous feedback to learners for their submissions to the SQL tutoring system. Learning and training are, however, continuous processes over a number of individual interactions. Supporting the student individually through observation and personalised feedback and recommendation is one of the most important benefits of computer-enhanced learning [14]. Feedback can be global or local [18], which we address through synchronous local feedback as part of corrections and global guidance and recommendations based on the student's overall performance.

We revisit correction, but from the interaction perspective under consideration of student model and feedback generation, and we introduce the recommendation feature.

5.1 Personalisation

Individual problems in the SQL environment are given in a suggested order. While personalization often addresses the navigation infrastructure (e.g. path selection) between these problems, our personalization focus is on content-based interaction in the form of feedback.

Students can, if desired, choose feedback levels the for correction and recommendation features. However, the system also provides a personalised feedback that considers the subject competency of the student. Feedback levels are ncreased if the student has difficulties; feedback is faded out, if the student becomes self-reliant and competent. The system automatically provides personalization. Although this can be overruled, the majority of students use the recommended automated feedback levels.

5.2 Recommendation Architecture

The recommendation architecture is presented in Fig. 5. Again, it serves to outline and structure the main functionality and interactions. In terms of ITS components (see Section 2.3), the pedagogy component is the predominant functionality provider here – implemented by the recommendation component. This component accesses student and expert model.

5.3 Learner Model – Profiling and Diagnosis

In order to provide a personalized learning experience, we propose a learner model with two central components:
- preferences determined by the student herself/himself – we only consider a simplified range of aspects in this investigation that are specific to the proposed feedback in terms of correction and recommendation.
- performance determined through observation of the student's interaction – essentially the degree of correctness and the types of errors a student has made over a period of time.

The preferences determine the quality and quantity of correction and recommendations. Students can choose from a number of predefined feedback and recommendation levels.

The performance diagnosis captures errors made in the categories syntax, semantics and pragmatics. In the semantics category, which we focus on here, a facetted schema of errors made by a student is applied. A performance entry in the learner model records errors made by a student. It consists of
- the problem that the student has tried to solve – a reference to the problem base
- an error classification in terms of
  - clauses as the computational perspective
  - elements as the object perspective
- the number of occurrences for each problem/error combination

These entries record individual errors and their categories to determine the student's strengths and weaknesses. This information can be used to recommend further study material to a student or to allow grading the overall performance.

5.4 Correction

The correction feature provides interactive synchronous feedback. The aim is to support formative and immediate learner assessment. We distinguish a number of predefined feedback levels that are part of the learner preferences:
- essentially, the quantity of feedback is determined where more feedback also means a better quality,
- the standard setting provides hints, i.e. does not reveal the solution, but aims to help the student to reassess her/his answer. Hints are weighted to reflect the different severities of individual errors.

Some of the individual levels for correction feedback are:

- Level 1: one SQL clause hint (highest weight)
- Level 2: one SQL clause hint (highest weight) and one SQL element hint (highest weight)
- Level 3: all SQL clause hints and all SQL element hints
- Level 4: further hints and links to relevant background material are added.

For our example from the previous section, for each level the following hints are added for levels 2 to 4:

- Level 2: there was an error in the select clause and there was an error with an attribute that was selected.
- Level 3: there was an error in the where clause; there was an error with a symbol that was used.
- Level 4: you need to include the following element: "colour … <> …" and try the following links – 1 – 2 – 3 - that provide background material for this type of problem.

The standard setting provides only one type of feedback – hints. Hint levels are determined only based on the preferences. In addition, the tutoring system supports scaffolding, i.e. a student support feature that aims at guiding the student towards sufficient self-reliance and expertise to solve problems in practice. The scaffolding uses three types of feedback in a strategy that extends the preferences model:

- hints are as usual advice to correct an error,
- partial solutions reveal critical elements of a solution,
- full solutions reveal the ideal solution.

These types are applied in a staged strategy. Initially only hints are given. If errors kept being made, the system switches to partial and then full solutions. Over time, the quality and quantity of feedback is reduced – called fading in scaffolding theory – in order to achieve self-reliance.

5.5 Recommendation

The recommendation feature provides interactive guidance over longer learning and training periods. Its aim is summative and accumulative learner assessment.

The recommendation feature provides a student with suggestions for further study. In case of this SQL tutor, the system proposes a number of exercises of the same kind as the ones the correction feature is based on.

The recommendation determination is based on the following factors:

- diagnosis-based, i.e. personalized considering individual student weaknesses extracted from the performance model,
- accumulated, i.e. based on weighted and ranked learner assessments considering the performance data from the student model,
- filtered, i.e. presentation based on preferences and scaffolding principles (recommendation strategy).

The recommendation algorithm is based on the following steps:

- Step 1: grouping of errors by error category, i.e. for each clause/element error pair
- Step 2: calculation of number of errors per error category
- Step 3: ranking of errors by number of errors per category
- Step 4: grading of overall performance based on the weight for each problem and the number and severity of errors made (errors in simple problems result in lower grades than errors in difficult problems) – summarized and averaged out over all problems attempted.

$$\left(\sum \frac{\textit{number of errors}}{\textit{problem weight}} \right) / \textit{number of problems}$$

  An error indicator is calculated – the higher the value, the more/severe errors have been made. The result is then normalized to a scale 0 … 100.
- Step 5: for each error type, ordered by number of errors), problems are retrieved from the problem repository that match the clause/error pairs of errors made. A cut-off point to limit the number of recommended problems can be defined as part of this level scheme

As for correction feedback, the student can choose between a number of predefined recommendation feedback levels that provide increasing amount and quality of recommendations.

- Level 1: one SQL clause/element diagnosis
- Level 2: all SQL clause/element diagnosis (graded)
- Level 3: all SQL clause/element diagnosis (graded) + menu of relevant further problems to practise

For instance, the following advice could be given by the tutor:

- Level 1: your greatest amount of errors involves the WHERE clause, using symbols.
- Level 2: you are having major problems with the WHERE clause; you are having minor problems with the SELECT clause.
- Level 3: try the following questions here (follow the link to further selected problems).

The virtual apprenticeship model, which we incorporated in this recommendation approach, assumes that the student takes a degree of responsibility for her/his learning experience. Therefore, recommendation only suggests areas that the student should revisit or pay particular attention to.

6 Discussion

We have introduced feedback techniques for correction and recommendation for a learning and training system that provides immediate and accumulative feedback to students who learn a computer language. A number of questions arise in this context:

- what are the difficulties of implementing such a system?
- is the system effective?
- can the correction and the diagnosis be used to support other tasks such as grading?
- can the approach be transferred to other computer languages?

We also discuss some related work in this section.

6.1 Accuracy

Accuracy is the crucial property of a correction system – whether it is used for feedback only or for grading. The degree of accuracy can be hampered by two factors in a syntax-driven approach such as ours. These factors relate to the two main activies decribed in Sections 4.5.3 and 4.5.4:

- Correctness determination. The possibility of having semantically equivalent solutions that are syntactically different can affect accuracy negatively. A complete set of equivalence rules need to be defined and applied.
- Error classification. The complexity of the language itself can affect accuracy: while correctness itself can still be easy to deal with, the quality of different error categories can be difficult to address in a weighting and ranking system for multiple errors.

In our experience with SQL, accuracy has been difficult to achieve in practice. The SQL query language is here of medium complexity – more complex than some diagrammatic notations, but much simpler than some programming languages that integrate different computational paradigms and mechanisms.

In our case, dealing with semantic equivalence has been a major problem from the implementation perspective. Two problems have arisen that concern the results of Section 4.5.3:

- the completeness of the set of equivalence rules: are all possible equivalences covered?
- the correctness of the rule application implementation: is the application of the rules correct in that the tutor implementation detects exactly those equivalences defined by the rules?

While both questions can theoretically be answered positively, we have not carried out full proofs for either question. A comparison with similar systems, such as the SQL-Tutor from the University of Canterbury [18] that uses a constraint base, shows that these rules or constraint bases are usually incrementally developed using an empirical approach. A validation of the application strategy – in our case the bottom-up strategy to apply equivalence rules to a syntax tree – is equally usually done empirically.

Most ITS, such as the SQL tutor by the University of Canterbury [19,21], use heuristics-based approaches. Query optimization in the SQL context [10] – or any other technique to determine the equivalence of language expressions – could provide an alternative solution. While in terms of accuracy, at least equally good results can be achieved, ITS need an approach that allows addressing specific cases explicitly, such as common errors as observed by an instructor [9].

6.2 Implementation and Evaluation

6.2.1 Context and Evaluation Method

We have implemented the described SQL tutor and it has been used the first time in the academic year 2004/05 as part of an undergraduate introduction to databases with about 100 students per year. Student attainment and student opinion are two central success criteria for learning support tools. An evaluation of student behaviour in the systems can also add valuable information in terms of system effectiveness.

We used a hybrid evaluation method based on traditional surveys and a new form of data mining. General information about student behaviour and usage of the system can be determined through web usage mining, which we have used to validate and complement the results of both the attainment evaluation and student survey. Although this evaluation often shows examples of just-in-time learning, we observed that this type of learning behaviour is complemented by long-term and pre-emptive use. The majority of usage occurred outside of the scheduled lab sessions, which demonstrates its value as a self-study tool following the apprenticeship philosophy.

6.2.2 Evaluation Results and Discussion

Student attainment is a central metric to determine the success of an e-learning system. However, the motivation, organization and usage of the system by the students are factors that determine acceptance and ultimately attainment [24]. We used the hybrid evaluation method to analyse student opinion and behaviour.

Student attainment is one of the factors that determine the effectiveness and success of the tutor. We did improve the online support system IDLE for the database module over many years regularly and have achieved an examination mark increase by 2% annually through improvements (excluding the SQL tutor introduction). While during the introduction of our tutoring system, all other factors remained constant, the relevant exam results have increased by 5.4% on the previous year, which is a significant improvement.

Student opinion is another crucial success criterion. Over 90% agreed, some strongly, that the tutoring system was a useful teaching and learning tool in its own right. The majority of students agreed or strongly agreed that the system is easy to use in general. Survey answers demonstrates that students accept virtual tutorials as equally suitable and effective as traditional tutorials. We have asked the students about their preference of delivery mode with respect to performance in exams. The opinion is split. This result shows the acceptance of virtual tutorials – virtual tutorials are at least as good as traditional ones – as a means to support one of the students' major objectives – good coursework and exam performance.

The feedback and personalised guidance features are seen as an important part of the tutoring system. Our survey results imply that the idea of providing both feedback and guidance has its merits. One critical aspect, however, is worth noting. Unintended higher levels of inaccuracy of the initial prototype have led to a more critical evaluation by students. Accuracy is the crucial property of the correction feature, even if it is not used for grading. Accuracy strongly affects the acceptance to the approach and the tool. Students who have encountered these inaccuracies felt discouraged to trust the system.

Students were asked to rate a number of statements about the SQL tutor. Students indicated that the most important aspect of the system is its focus on active skill-oriented learning rather than a passive lecture-based approach. The second most important aspect is that the system is always available and enables self-paced learning, which would otherwise only have been possible during supervised lab times.

6.3 Further Applications

Automated grading is possibly the application that seems natural. Whether a system such as the proposed one can be used for grading depends essentially on the accuracy of the correction feature.

In general, two grading approaches can be distinguished:

- coarse-granular: the system distinguishes only between correct and incorrect. This eliminates the complexity of error weighting and categorization of multiple errors.
- fine-grained: the system attempts to provide a similarity measure based on an error weighting. This combines the complexity of correctness determination with the one from error classification.

Grading could be implemented by linking a severity level to errors and linking an importance level to elements in the clause. Students could be given full marks initially, which are decreased in varying amounts by the correction model based on the severity of errors and the absence of vital elements.

Our experience shows that even minor inaccuracies in the correction can lead to acceptance problems, which we have experienced with our first prototype. However, we believe that through incremental improvements, a correction system can reach a degree of accuracy that is similar to the accuracy of the educator as mechanical processing, in particular since its automated nature removes other, human-typical inaccuracy reasons [29].

6.4 Transferability

Our aim was to present a generic approach that can be applied to other computer languages. This can potentially include diagrammatic languages as well as more complex programming languages. We outline prerequisites and identify possible difficulties. We illustrate the potential using an example.

While, as our discussion of accuracy shows, the transfer of the approach to complex programming, specification or modeling languages might require some implementation effort to achieve at least acceptance (and might initially rule out reliable grading), languages of similar or smaller complexity can be addressed from a practical point of view. Within the database context, we have started to explore the application of these principles to Entity-Relationship diagrams as the notation. In a similar way, problems and ideal solutions can be prepared and equivalence rules defined by an instructor and/or a knowledge engineer.

Formally, a language representation in terms of grammars, abstract and simplified abstract syntax trees and semantic equivalence rules based on the grammar are the only prerequisite. Suitably formulated grammars that allow the effective parsing of a language expression are, of course, required. Once a language is mapped onto this infrastructure, our approach can be applied, only limited by the scalability of the accuracy problem.

In general, research on notions of equivalence in the language under consideration – such as query optimization for query languages or bismilarity for process languages – is beneficial to the implementation work as it can provide the formal foundations of correction.

6.5 Related Work

Our system enables activity-based learning and training based on an intelligent tutoring system (ITS) architecture. We introduce three systems that provide personalisation and feedback for SQL tutoring. These systems are similar in their aims and interfaces, but they each have differing architectures and methods of correction.

- SQLator [30] corrects a student submission by equating it with the corresponding English question – the authors describe this as evaluating the student submission rather than correcting it. The system's equivalence engine judges if the SQL answer of the student correctly corresponds to the given English question, without actually executing the query at correction stage.
- Acharya [4] uses a three-step process – pre-processing, atom processing and truth table processing – to correct the student's answer. It is similar to our in that it is syntax-driven. The Acharya process assumes that the sets of literal atoms in two language expressions are the same. The process fails if one of the expressions is made up of more atoms than the other. This can actually result in accuracy problems. In its current implementation, only the where clause is analysed. The truth table technique to determine semantic equivalence is specific to the where clause, i.e. other techniques would be required to address other SQL clauses.
- SQL-Tutor [21] uses constraint-based reasoning to correct answers submitted by the student. The constraints, which are stored in a constraint base, deal with syntax errors as well as semantic errors, as the student's proposed solution is not actually executed. The system checks each submitted query for relevant constraints that might have been violated. This method of correction can yield a high level of accuracy, depending on the extent of the constraint base – which re-iterates our discussion of accuracy and the effort needed to achieve it. In order to provide for the large range of possible errors, a large constraints based has been developed incrementally over a significant period of time.

Brass and Goldberg [7] have developed a related approach to correcting SQL queries based on a correction tool that gives feedback on general queries without an ideal solution using heuristics about the general consistency of the query. While this would provide a useful

complement to a training or development environment for databases, it does not provide an adequate degree of feedback in an automated tutoring setting.

The first three systems offer some form of scaffolding and feedback to the student.

- SQLator's automatic feedback consists of an error flag signaling the correctness of a student answer. Asynchronous feedback is offered by allowing academic staff to email or post messages to address submissions. A synchronous hint-based feedback system, which we consider essential for automated tutoring, is proposed as an extension [30].
- Acharya provides feedback in the form of error flagging and hints. Hints are comprised of text and links. Only one hint is displayed at a time. The system does not offer guidance based on the student's performance.
- SQL-Tutor provides advanced feedback, offering both hints and partial solutions. There are five levels of feedback ranging from a simple correctness indication to offering the complete solution. Feedback is in text form only; relevant links to background material that would put errors into context are not offered. There are no accumulative recommendations.

Both Acharya and SQL-Tutor suggest the next question for the student to attempt. The reasons for this system recommendation are, however, not made explicit to the student as in our guidance component and the student is not given a choice of recommended questions.

7 Conclusions

Automated tutoring has become an accepted method of instruction. Students reach a higher level of understanding when being actively engaged in learning and training processes. Our automated tutoring system provides a realistic training environment for database programming. Automated tutoring is time and location independent. While generally a beneficial characteristic, scaffolding is here a necessary feature for automated tutoring in this context. Feedback is a classic and central example of scaffolding. Correcting, providing feedback, presenting recommended questions, etc. is part of the scaffolding needed for knowledge-level interactions between student and tutor.

We have demonstrated that this scaffolding can be implemented for computer language learning and training based on two knowledge and data representation techniques:

- language grammars as the structuring principle that defines the learning content and its underlying knowledge,
- syntax trees and their abstractions as data structures that capture problem-based knowledge that is communicated between student and system.

One of our objectives here was to introduce techniques and to demonstrate:

- the potential of advanced tutoring for a computer language based on a pattern matching approach to automated correction,
- the benefits of integrated feedback and personalised guidance based on pattern-based correction.

Course subjects focusing on computer languages lend themselves to automated tutoring as their structure makes them easy to analyse. The student can make submissions to a tutoring system and receive results automatically corrected result. The presented correction approach using grammar-based pattern matching can be transferred to similar learning and training area where computer-processable languages, both textual and diagrammatic, can be processed automatically based on techniques derived from an explicitly formulated grammar.

Some difficulties need to be addressed in the implementation of automated tutoring systems. A system's accuracy and the student's trust level, which is affected by accuracy, are important for its success. Designing and implementing a flawless correction method is, however, a challenge. However, our experience shows that sufficient accuracy can be achieved for a language of the complexity of SQL in order to make automated tutoring acceptable for students as a learning support tool and for educators to even consider it to support grading.

A note on the human and automated tutors shall conclude this investigation. An automated system can never fully replace human tutoring in terms of quality. This can only be alleviated to some extent by providing a useful system that greatly improves accessibility and availability. Personalised guidance complements immediate correction and feedback – and should be present in any tutoring system.

References

[1] S. Abiteboul, R. Hull and V. Vianu. "*Foundations of Databases*". Addison-Wesley. 1994.

[2] A.V. Aho, M.S. Lam, R. Sethi and J.D. Ullman. "*Compilers: Principles, Techniques, and Tools*" (2nd Edition). Addison-Wesley. 2007.

[3] A.V. Aho, J.D. Ullman and J.E. Hopcroft. "*Data Structures and Algorithms*". Addison-Wesley, Series in Computer Science and Information Processing. 1983.

[4] S. Bhagat, L. Bhagat, J. Kavalan and M. Sasikumar. Acharya: "An intelligent tutoring environment for learning SQL". *Proceedings of Vidyakash 2002 – International Conference on Online Learning*. 2002.

[5] J. Beck, M. Stern and E. Haugsjaa. "Applications of AI in education". *ACM Crossroads*, The Student Journal of the association for Computing Machinery, 3 (1). 1996.

[6] T. Boyle. "Design for multimedia learning". Prentice Hall Europe. 1997.

[7] S. Brass and C. Goldberg. "Semantic errors in SQL queries: A complete list". *Journal of Systems and Software* 79:630-644. 2006.

[8] P. Brusilovsky. "Adaptive Hypermedia: From Intelligent Tutoring Systems to Web-based education". *Proceedings of 5th International Conference on Intelligent Tutoring Systems, ITS 2000*. Springer Verlag. pp. 1-7. 2000.

[9] P. Brusilovsky, J. Knapp, and J. Gamper. "Supporting teachers as content authors in intelligent educational systems". *International Journal of Knowledge and Learning*, 2 (3/4), 191-215. 2006.

[10] S. Chaudhuri. "An overview of query optimization in relational systems". *Proceedings of the Seventeenth ACM Symposium*

on Principles of Database Systems PODS'98. ACM, pp. 34-43. 1998.

[11] C.-Y. Chou, T.-W. Chan and C.-J. Lin. "Redefining the learning companion: the past, present, and future of educational agents". *Computers & Education*, 40 (3), pp. 255-26. 2002.

[12] A. Collins. "Cognitive apprenticeship and instructional technology". Technical Report No. 6899. BBN Labs Inc., Cambridge, MA, USA. 1998.

[13] A. Collins, J.S. Brown and A. Holum. "Cognitive Apprenticeship: Making thinking visible". *American Educator*, Winter edition. 1991.

[14] P. De Bra, D. Smits and N. Stash. "Creating and Delivering Adaptive Courses with AHA!". *Proceedings of the first European Conference on Technology Enhanced Learning, EC-TEL 2006*, Springer LNCS 4227, pp. 21-33. 2006.

[15] D. Heaney and C. Daly. "Mass production of individual feedback". *Proceedings of Intl. Conference on Innovation and Technology in Computer Science Education ITiCSE'04*. ACM Press. pp. 117-121. 2004.

[16] J. Herrington and R. Oliver. "An instructional design framework for authentic learning environments". *Educational Technology Research and Development*, 48 (3), pp. 23-48. 2000.

[17] C. Kenny. "Automated Tutoring for a Database Skills Training Environment". M.Sc. Thesis. Dublin City University, School of Computing. 2006.

[18] C. Kenny and C. Pahl. "Automated tutoring for a database skills training environment". *Proceedings of ACM Special Interest Group on Computer Science Education SIGCSE Symposium 2005*. ACM Press. pp. 58-62. 2005.

[19] M. Mathews and A. Mitrovic. "Investigating the Effectiveness of Problem Templates on Learning in Intelligent Tutoring Systems". Proc. 13th Int. Conf. Artificial Intelligence in Education AIED 2007, pp. 611-613. 2007.

[20] E. Melis and C. Ullrich. "Local and global feedback". *Proceedings of AIED2003, 11th International Conference in Artificial Intelligence in Education*. 2003.

[21] A. Mitrovic, B. Martin and P. Suraweera, "Intelligent Tutors for All: The Constraint-Based Approach". *IEEE Intelligent Systems*, 22(4), pp. 38-45. 2007.

[22] T. Murray. "Authoring Intelligent Tutoring Systems: An analysis of the state of the art". *International Journal of Artificial Intelligence in Education*, 10, pp. 98-129. 1999.

[23] S. Murray, J. Ryan and C. Pahl. "A tool-mediated Cognitive Apprenticeship approach for a computer engineering course". *Proceedings of International Conference on Advanced Learning Technologies ICALT2003*. IEEE Press. pp. 2-6. 2003.

[24] C. Pahl. "Behaviour analysis for Web-mediated active learning". *International Journal of Web-Based Learning and Teaching Technologies* 1(3), pp. 45-55. 2007.

[25] C. Pahl, R. Barrett and C. Kenny. "Supporting active database learning and training through interactive multimedia". *Proceedings of the 9th Annual Conference on Innovation and Technology in Computer Science Education ITiCSE'04*, ACM Press. 2004.

[26] R. Ramakrishnan and J. Gehrke. "Database management systems". McGraw Hill. 2003.

[27] A. Ravenscroft, K. Tait and I. Hughes. "Beyond the media: Knowledge level interaction and guided integration for CBL systems". *Computers in Education*, 30 (1/2), pp. 49-56. 1998.

[28] P. Reisner. "*Human factors studies of database query languages: a survey and assessment*". ACM Computing Surveys 13, pp. 13–31. 1981.

[29] A. Rizzo, S. Bagnara and M. Visciola. "*Human error detection processes*". International Journal of Man–Machine Studies 27, pp. 555–570. 1987.

[30] S. Sadiq, M. Orlowska, W. Sadiq, and J. Lin. SQLator – an online SQL learning workbench. *Proceedings of Intl. Conference on Innovation and Technology in Computer Science Education ITiCSE'04*, June 2004. pp. 223-227. ACM Press. 2004.

[31] J. Stephenson (Ed.). "Teaching and learning online". Kogan Page. London. 2001.

[32] R.H. Stottler and M. Vinkavich. "Tactical Action Officer Intelligent Tutoring System (TAO ITS)". *Proceedings of the 2000 Interservice/Industry Training, Simulation and Education Conference (I/ITSEC-2000)*. 2000.

[33] K. Winnips. "Scaffolding-by-design as a model for online learner support". Ph.D. thesis, Faculty of Educational Science and Technology, University of Twente, Netherlands. 2001.

Claus Pahl (M.Sc., Ph.D.) is a Senior Lecturer and the leader of the Web and Software Engineering research group at Dublin City University, which focuses on Web technologies and e-learning applications in particular. Claus has published more than 150 papers including a wide range of journal articles, book chapters, and conference contributions on e-learning. He is on the editorial board of the International Journal on E-Learning and the International Journal of Technology-Enhanced Learning and is a regular reviewer for journals and conferences in the area of software, Web, and learning technologies and their applications. He is a member of the IEEE and the IEEE Computer Society.

Claire Kenny (B.Sc., M.Sc.) is a Research Assistant at Dublin City University, currently involved in an EU-supported learning technology project on learning object development and reuse. Claire has recently completed her M.Sc. by research on a topic in intelligent tutoring systems. She has been involved in the development of content and supporting infrastructure for technology-enhanced active learning for several years.