Learning SQL with Artificial Intelligent Aided Approach

Tadej Matek, Aljaž Zrnec, and Dejan Lavbič

***Abstract*—Efficient data manipulation and retrieval is a fundamental part of many business processes in the majority of todays' companies. SQL, as a standard, is widely adopted and well accepted in this area. Students who set out to learn SQL frequently face difficulties. The learning process is to some extent inefficient, as the student's knowledge is afterwards often inadequate. Several computer-aided systems have been developed to alleviate the problem. However, most of them are static and rigid, because the system's knowledge is encoded manually. We propose a new system based on past attempts and solutions to SQL exercises. The proposed system is flexible and dynamic, as it adapts to the individual student and requires minimal intervention from domain experts. We show that the system is beneficial, in particular to students with low prior knowledge.**

***Index Terms*—Intelligent tutoring systems, SQL learning, Markov Decision Processes, adaptive hint generation.**

I. Introduction

The majority of today's companies depend on efficient data storage, retrieval and manipulation. In the past century a standard for data manipulation has emerged, namely the Structured Query Language (SQL). Even though alternatives, such as NoSQL and NewSQL, are also beginning to gain traction, most of computing systems nowadays actively use SQL. The need for well-qualified personnel, proficient in SQL, is great. Students obtain knowledge of SQL either through an undergraduate database course, online courses or in other technical institutions. The students however, after completing a course in SQL, do not necessarily posses the proper knowledge or their knowledge is inadequate [1]. Typically, students misinterpret the fundamental concepts of the language and in general have difficulties in formulating correct and more importantly, efficient queries. Common misconceptions include concepts such as data aggregation, joins, filtering using predicates etc. Additionally, the students have difficulties memorizing database schemas, table and/or attribute names while being graded [2]. It was recognized by Prior *et al*. in their study, that students are unable to visualize the result of their written query, unless there is an option to execute their query against the database. Consequentially several SQL teaching systems now include a visualization of the database schema and also attribute and table names to relieve the students during grading. Such systems frequently offer the option of testing one's query to receive instant feedback. The core of the problem however, is not in the difficulties presented by the grading process but by the complexity of the language itself.

Manuscript received August 13, 2016; revised January 5, 2017.

Tadej Matek is with the Faculty of Computer and Information Science, University of Ljubljana, Slovenia.

Aljaž Zrnec and Dejan Lavbič are with the Laboratory for Data Technologies, Slovenia (e-mail: Dejan.Lavbic@fri.uni-lj.si).

The learning process is structured in a way to test select concepts of the language. The complexity of the exercises is higher than expected, from the student's perspective, unless that student has had extensive practice beforehand. Several computer-aided systems were developed to alleviate this problem, with intelligent tutoring systems (ITS) being the most successful [3]-[5]. Such systems employ hints to aid the educational process. ITS systems are also being used on SQL domain, but currently lack dynamic and adaptive behaviour. Due to this, domain experts are required in order to manually encode the knowledge used by the system, which is then static. The time and cost of these systems is high. Furthermore, the current systems lack the ability to adapt to an individual student.

We propose a new system, which is capable of generating hints for various states of SQL exercise-solving process and requires minimal intervention from the experts, as all hints are generated using past student attempts. Our system is adaptive in the sense that it adapts to the current state of the student and offers a specific hint. We perform evaluation of our system in an actual learning environment to measure the impact and usefulness of the hints.

The rest of this paper is organised as follows. The next section provides a brief overview of computer-aided education in general, intelligent tutoring systems and approaches for learning SQL. In Section III we provide an extensive description of the proposed system with system's architecture and the process of hint generation. In Section IV we perform an empirical evaluation of the proposed system on a group of 93 participants with diverse prior knowledge of SQL. We conclude and provide future directions for improving our research in Section V.

II. Related Work

Although in early Computer Aided Instruction (CAI) systems students may have had some influence on navigation through the curriculum, they all received the same contents [3], [6], [7]. In the later CAI systems branching provided different responses to a student's answer, depending on what student's response was [6], [8]. This kind of CAI systems possessed no domain knowledge, meaning that every feedback had to be provided by experts manually. Usually the feedback was limited to right/wrong answers and sometimes to presentation of a correct answer.

With CAI systems, modest gains can be reached in contrast to classroom learning, but one-to-one way of tutoring can not be achieved [9]. That encouraged researchers to improve computer-based teaching environments to more closely imitate human tutors, resulting in emergence of Intelligent Tutoring Systems (ITS) [4], [10], which adapt to user's individual needs. Such systems recommend educational activities and deliver individual feedback regarding the

doi: 10.18178/ijiet.2017.7.11.976

student's profile, specified by the student's knowledge or activities within the course.

The most acclaimed ITS systems evolved through the history are cognitive tutors [11], [12] followed by approaches, based on constraint-based modelling [3], [13], [14] or the principle of constructing student models with machine learning techniques [6], [15].

Cognitive tutors base on ACT* theory [16] and are intended for solving procedural problems, where the problem domain is modelled as a set of production rules. A system monitors the student while solving a problem and when he/she deviates from the predicted path, it offers a hint automatically. Cognitive tutors are too restrictive and as such not suitable for all problem domains. They are also hard to build, because advanced AI programming skills are required to design them. For example, to define necessary rules for SQL domain, it might take the experts more than one year [17], which represents a serious obstacle.

Constraint-based modelling (CBM) is used in the second approach to building ITS systems [3], [18]-[20]. Here, the knowledge of target domain is represented as a collection of constraints. If all constraints are satisfied, the system considers solution as correct, but if constraints are violated they become targets for instruction. If specific constraint is violated, the CBM system provides a feedback associated with that constraint. An advantage of CBM system is that no additional study of student's errors is required for its implementation. An important set of errors is implicitly defined by aforementioned constraints. SQL-Tutor [3] is an important representative of CBM system.

A significant amount of time is needed to define rules and constraints for cognitive tutors and CBM systems to become useful [14], [21], [22]. To some extent, AI methods can be used to automatically generate set of rules and constraints, if there is enough data about specific domain to learn from [13], [22]. In this way domain experts, who are not skilled programmers can produce contents for ITS systems. Domain experts performing programming by demonstration is an approach used to describe particular problem domain, while the system builds proper program constructs independently.

Data logged during student learning have also become important, since sufficient amount of data enables information extraction for further usage in learning systems. The approach is used to build knowledge base for specific domain, using former student solutions. For instance, Hint factory [5] is an intelligent system that uses student data to build Markov Decision Process (MDP), which represents all approaches to a particular problem and uses MDP for hint generation.

Building ITS systems has always been expensive and time-consuming process, where many different experts with adequate programming knowledge have been involved [5], [6], [13], [23]. We would like to find a balance among the complexity of building such systems, invested time and their price. We also strive for the system to work successfully and to perform an effective learning process. Consequently, we have to analyze the disadvantages of previously presented approaches to find suitable solution.

Cognitive tutors are rather intrusive from the student's perspective, because they constantly correct user entries and warn us if we turn off the planned path. Since the domain model is made of procedural rules and if there are alternative paths to problem solution, more rules are needed, which requires more time and is related to higher costs. That usually leads to definition of rules for one alternative only [8]. In case of CBM systems, the generality of used constraints represents an issue reflecting in misleading feedback. Unlike the cognitive tutors, CBM system does not possess any information about problem-solving procedure and it only checks matching the solution to general constraints.

Cognitive tutors and CBM systems are based on static approach for building problem domain. Only experts who precisely understand the domain can build these systems. With AI methods problem domain can be built dynamically, which eliminates the need for domain expert and allows teachers without the understanding of how these systems work, to build intelligent systems. In case of using domain model, based on historical data, we can map student's solution to historical data of their colleagues. The complexity of such solution still remains high.

Our system is based on past student attempts (historical data) at solving SQL related exercises. We used the information within the data itself to remove the need of an expert to encode the solutions manually. One of the advantages of our system is automatic building of knowledge base with no required human intervention. The hints being given to students are based on correct solutions of their colleagues from previous study years. Additionally, the system can adapt to student's path of solving the problem and offers a hint precisely targeted to student's current solution, rather than offer a general, undirected hint. AI represents a vital part of the system and its task is to perform state exploration until the best state is found and construction of the knowledge base using past attempts. Comprehensive description of the system internals is provided in the next section.

III. Proposed System

The proposed system supports both the student’s and teacher’s aspect. The basic architecture of the system is depicted in Fig. 1 and covers the teacher’s aspect, the student’s aspect and the architecture of the recommender system. An instructor (teacher) is able to prepare assignments by specifying assignment instructions, the ideal solutions for the exercise and evaluation rules, which determine how is the grading performed. One can observe the individual components of the recommender system involved in each process (for example in assignment formulation process the following components of the recommender system are used sequentially: W1, A2, D1).

The student’s perspective allows the use of hints to aid the exercise-solving process. Students may formulate and adapt their query, execute it to receive feedback or request a hint. When requesting a hint, an adapted query is returned, according to the current student’s formulated query. Student may use the hint or ignore it altogether. At the end of the process students submit their solution as final.

The basis of our system are past attempts of students (SQL queries) from previous generations. The historical data were collected during years 2012 and 2014 in a process of evaluating student’s SQL skills during the “Introduction to databases” course at University of Ljubljana, Faculty of

Computer and Information Science. Overall the solution pool contained over 30.000 entries spread across approximately 60 exercises and 2 different database schemas (including the well-known *Northwind* database). Our data contain the necessary information to create a model, which adapts to a student and offers a specific hint rather than a general one. The system is based on Markov decision process (MDP), previously used for hint generation by Barnes *et al*. [5], but adapted for SQL domain. MDP is defined as a tuple

$$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle, \tag{1}$$

where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ a finite set of actions connecting the states, $\mathcal{P}$ a matrix of transition probabilities, $\mathcal{R}$ a reward function and γ a discount factor. The behaviour of the agent in any of the states is stochastic, as matrix $\mathcal{P}$ defines the likelihood of an agent choosing a specific action and actually reaching the intended destination. Actions therefore have multiple destinations, which are nondeterministic. The reward function allows us to rank the states according to some criterion, so that the agent can avoid low reward states, or even states with negative reward (punishment), and rather focus on reaching high reward, desired states.

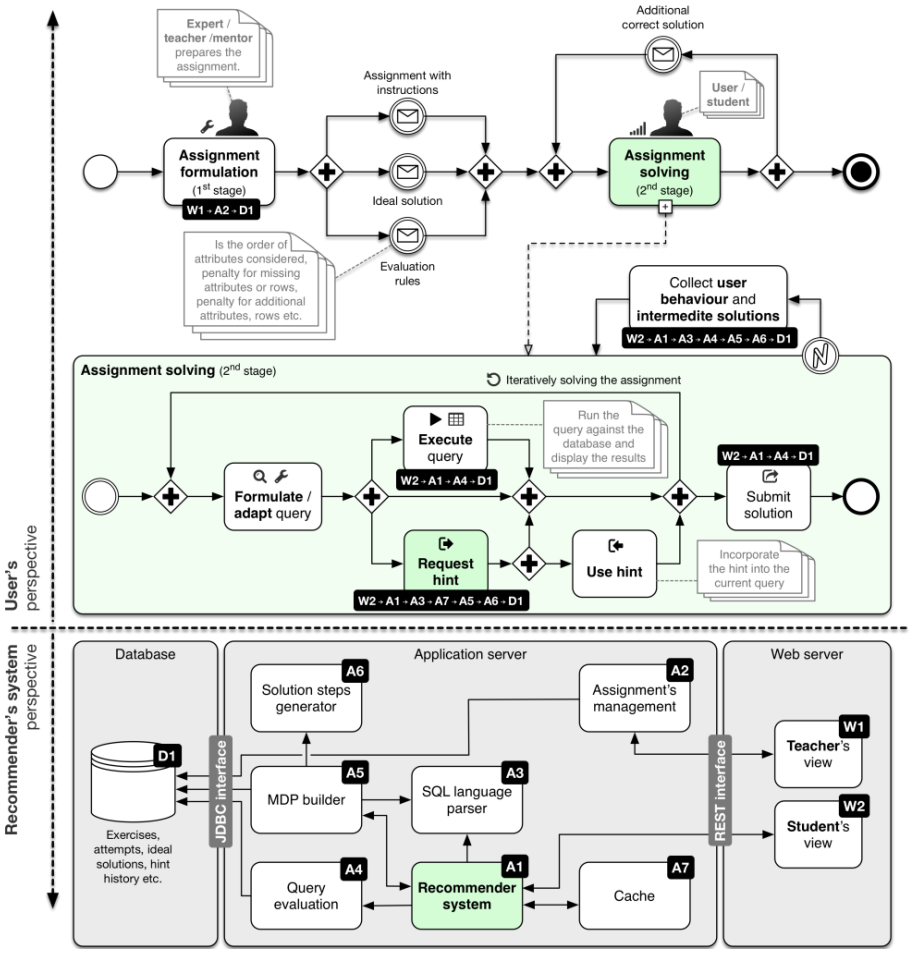


Fig. 1. Architecture of the proposed system.

The agent's behaviour in such dynamic system is defined by a policy π, which is essentially a mapping of states to actions and determines which action the agent should choose in every state it reaches. A value function is defined as

$$V^{\pi}(s) = \sum_{s'} P_{ss'}(\mathcal{R}_s + \gamma V^{\pi}(s')). \tag{2}$$

We make use of value iteration to iteratively update rewards of all states until a global maximum of the value function is reached:

$$V^{*}_{i+1}(s) = \max_{a \in \mathcal{A}}(\mathcal{R}_s + \gamma \sum_{s'} P^{a}_{ss'} V^{*}_{i}(s')) \tag{3}$$

The discount factor allows us to give priority to either short-term or long-term rewards. In our case we prefer the long-term rewards.

We do not directly use SQL queries found in historical data as states in MDPs. We first transform the text form of the queries into a tree structure using ANTLR parser [24]. An example of such tree structure can be seen in Fig. 2. The queries alone do not suffice to produce valuable hints, as they do not contain the order of query construction (i.e. how the students arrived at the final solution). Since neither our historical data contain such information, we were forced to generate it using merely the final solutions. The assumption we take is that the students construct their query in order of the general sections of the query. A section is any major part of the query such as SELECT clause, FROM clause etc. Such assumption is unlikely, however proves sufficiently useful. The sections assumption allows us to perform a depth-first tree traversal, stopping at right-most leaves and

producing a copy of the tree for each such leaf. This yields a forest of all the (possible) solution steps that lead to the final query, as can be seen in Fig. 2.

It is the generated individual solution steps, such as in Fig. 2., which are then mapped to states in MDP graph. Because each solution step is a prefix of the next solution step, in terms of query text representation, we add actions among consecutive solution steps (states). For a given exercise, we merge all possible solution paths (which consist of solution steps) into an MDP graph, while making sure there are no duplicate states. Probability matrix is determined as a relative frequency of students who moved from one state to another, given historical data. Rewards for final states (final solution steps) are determined using an existing query evaluation component. The component evaluates a query by comparing the result set of the student's solution with the result set of an ideal (instructor's) solution. Points are then deducted if rows or columns are missing, if the order of rows is incorrect etc. A final state receives a high reward if its query score is above 95%. Otherwise the state receives a negative reward. The end result is a collective knowledge base for a single exercise according to all the students, which made an attempt at solving this exercise.

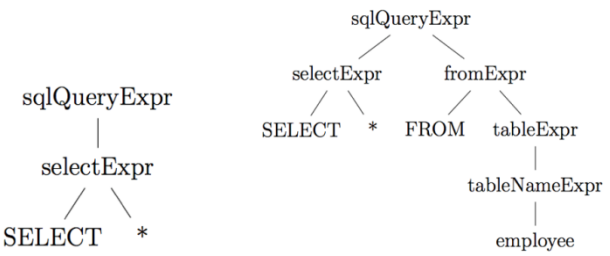


Fig. 2. An example of solution steps construction.

The hint generation process accepts the current student's query in order to match it with one of the states in the MDP graph. The system finds a corresponding MDP for the exercise first. We cache the MDPs for all exercises to avoid the overhead of database I/O. We parse the student's query in a similar way as the historical data queries, to produce a tree structure, which is then matched to one of the states in MDP graph. A tree distance criterion is used for matching, namely the Zhang-Shasha algorithm [25]. Once a matching state is found, the neighbouring state with the highest reward is offered as a hint (the query in the next state is converted back to a text representation). We also add backward actions to allow returning from an incorrect MDP branch. This can happen if the reward of taking the backward action is higher than the reward of taking any forward action. In such cases, the system returns from the subtree, which represents an incorrect solution, to the first common ancestor state of both the incorrect subtree and an alternative correct subtree. Skipping the entire incorrect subtree is necessary in order to ensure that hints are progressive. The system is also seeded using ideal solutions from instructors. This partially solves the cold-start problem of new exercises, which do not yet have historical data available for hints, as it allows the students to receive hints leading them to one of the ideal solutions.

An example of hint construction is visible in Table I. The first row of the table corresponds to a scenario, where the student is located in an incorrect MDP branch. Observe, that the system does not direct the student towards the ideal solution (one of the ideal solutions actually), but proposes an alternative MDP branch, which eventually leads to the correct solution. The alternative MDP branch was constructed by another student's solution from the previous generation. The last hint in the first row demonstrates that nested queries are also supported. The second row of the table corresponds to a scenario, where the student's solution is partially correct, yet the student fails to continue. As one can observe, the student forgot to include the *department* table, which is what the hint corrects. A visual example of how the hints are presented can be seen in Fig. 3.

TABLE I: EXAMPLE OF CONSTRUCTED HINTS FOR SPECIFIC EXERCISES, GIVEN STUDENT'S SOLUTION

Task description	**Ideal solution**	**Student's solution**	**Hints**
Return the number of employees in department 'SALES'	**SELECT** COUNT(*) **FROM** employee, department **WHERE** employee.dept_ID = department.dept_ID AND department.name = 'SALES'	**SELECT** * **FROM** department	**SELECT** COUNT(*) **FROM** department **WHERE** dept_ID **SELECT** COUNT(*) **FROM** department **WHERE** dept_ID IN (SELECT dept_ID)
Return the number of employees in region 'DALLAS'	**SELECT** COUNT(*) **FROM** employee e, department d, location l **WHERE** e.dept_ID = d.dept_ID AND d.loc_ID = l.loc_ID AND region = 'DALLAS' **GROUP BY** region	**SELECT** COUNT(e.emp_ID) **FROM** employee e, location l **WHERE** region = 'DALLAS'	**SELECT** COUNT(e.emp_ID) **FROM** employee e, location l, department d

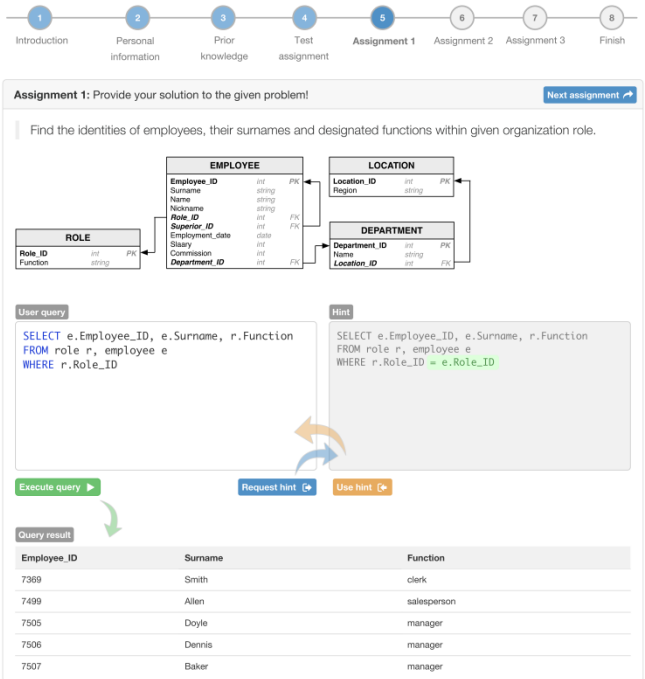


Fig. 3. Web interface of the exercise-solving simulation.

IV. EVALUATION

We evaluated the recommender system on a group of 93

participants, where each of them completed three assignments. Before conducting the experiment participants provided information about their prior knowledge – SQL proficiency level and years of experience in SQL. The obtained sample of participants had nearly normal distribution of their self-reported prior knowledge.

Every participant was required to provide a solution to three randomly selected SQL assignments that were classified into a category based on the difficulty level. Every participant was randomly allocated one easy, one moderate and one difficult assignment.

As Fig. 3 depicts participants entered a query into user's query box and interactively evaluated the results. When they were unable to continue, they requested a hint from the recommender system, which was then displayed next to the user query with indicated adaptations of the current query.

Every participant could request unlimited number of hints per assignments. To discourage participants to excessive use of hints or even solving the complete assignment with hints only, a small score penalty (inversely proportional to assignment complexity) was introduced for hint employment, which was clearly introduced to the participants before starting the evaluation.

To evaluate the performance of proposed recommender system we cluster participants into four groups, based on **hints employment** (no hints vs. using hints) and **SQL proficiency level** (low vs. high prior knowledge).

With evaluation of the proposed recommender system we focus mainly on the distance from the correct solution $dist_{sol}$ in a given time (number of steps to accomplish the correct solution). We define a linear association

$$\hat{dist}_{sol} = \alpha + \beta_{type} \cdot \hat{t}_{elapsed} \tag{4}$$

where type $\in$ (all, pre_first_hint, post_first_hint, after_hint_avg). The following Table II depicts aggregated mean results per predefined four clusters. When interpreting the results we strive to obtain as negative β_{type} values as possible, which indicate rapid advancement towards correct solution (e.g. minimize distance to correct solution over time) and by doing that, measure the effect of hint employment.

TABLE II: CLUSTER ANALYSIS RESULTS

hints employment	prior knowledge	β_{all}	$\beta_{pre_first_hint}$	$\beta_{post_first_hint}$	$\beta_{after_hint_avg}$
no hints	low pk	-1.12	-	-	-
no hints	high pk	-0.92	-	-	-
using hints	low pk	-0.61	-0.75	-1.61	-9.38
using hints	high pk	-0.89	-1.82	-1.63	-5.94

To further investigate the differences between four groups of participants, Fig. 4 depicts the most representational participant of every cluster group. The figure depicts the timeline of representational participant solving the assignment with every recorded action, where hint employment is highlighted (blue dots). Additionally, three regression lines and coefficients are also depicted, based on the following filtering: 1) β_{all} for all actions, 2) $\beta_{pre_first_hint}$ for actions before first hint employment and 3) $\beta_{post_first_hint}$ for actions after the first hint employment.

When considering groups of participants not employing hints we can observe that participants with high domain knowledge without major fluctuations reach the correct final solution in contrast to participants with low domain knowledge who experience some variation in terms of distance to correct solution $\hat{dist}_{sol}$.

When further observing participants that employed hints, we can conclude that proposed recommender system provide beneficial results as the impact of hint employment is evident. The $\beta_{after_hint_avg}$ is significantly lower than β_{all} for both subgroups (low and high prior knowledge). There is still a distinct difference between low and high prior knowledge subgroups. Fig. 4 depicts that participants with low domain knowledge will employ hint when they don't know how to proceed to the correct solution and then afterwards they experience the boost in terms of rapid decline of a distance to correct solution $\hat{dist}_{sol}$.

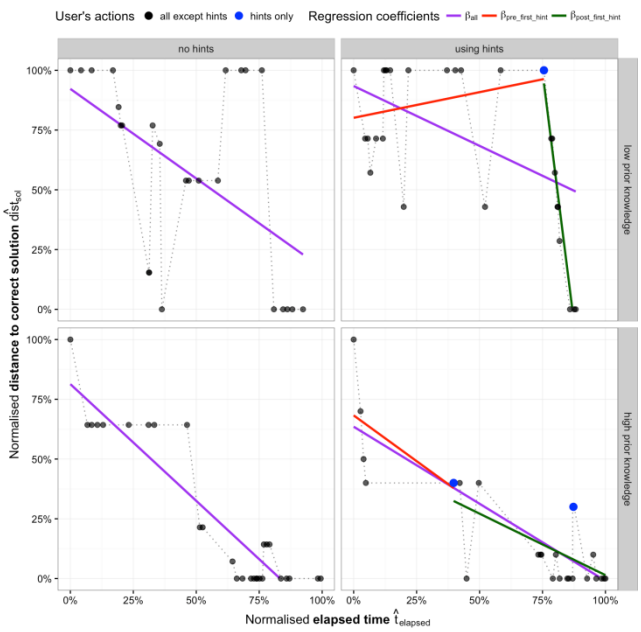


Fig. 4. Individual's cluster user timeline.

V. CONCLUSION

We have presented a new system, which was used to assist students during SQL learning process. The system makes extensive use of historical data, previous student attempts at similar SQL exercises. The proposed system is adaptive in the sense that it adapts to the current state of the student and offers a specific hint. Minimal intervention from the experts ensures that the system can be used and deployed by most people, without having to manually enter an array of production rules. In cases when there are no historical data for a specific exercise, the instructor can speed up hint generation by specifying ideal solutions. Furthermore, any new solution submitted by a student, can be included automatically in the system's knowledgebase.

The system's performance was evaluated in an actual learning environment with 93 participants of diverse prior knowledge. As expected the system is most beneficial to students with low prior knowledge. The hints turn out to be well accepted as the distance to correct solution drops singificantly after the employed hint and then remains relatively stable. The goal of the hints has a broader intent than to merely improve the score of the students. When a student does not know how to proceed, a hint may be

requested, which in turn may propose an alternative solution path. The students explore the alternative paths, performing errors while doing so, but also improving their skill set. This is also why the distance to correct solution for some students, after receiving a hint, is increased in the subsequent steps.

REFERENCES

[1] J. C. Prior and R. Lister, "The backwash effect on SQL skills grading," *ACM SIGCSE Bulletin*, 2004, vol. 36, no. 3, pp. 32-36.

[2] A. Mitrovic, "Learning SQL with a computerized tutor," *ACM SIGCSE Bulletin*, 1998, vol. 30, no. 1, pp. 307-311.

[3] A. Mitrovic, S. Ohlsson, and D. K. Barrow, "The effect of positive feedback in a constraint-based intelligent tutoring system," *Computers & Education*, 2013, vol. 60, no. 1, pp. 264-272.

[4] S. Schiaffino, P. Garcia, and A. Amandi, "ETeacher: Providing personalized assistance to e-learning students," *Computers & Education*, 2008, vol. 51, no. 4, pp. 1744-1754.

[5] T. M. Barnes, J. C. Stamper, L. Lehman, and M. Croy, "A pilot study on logic proof tutoring using hints generated from historical student data," in *Proc. Educational Data Mining*, Montreal, Quebec, Canada, 2008.

[6] G. Stein, A. J. Gonzalez, and C. Barham, 'Machines that learn and teach seamlessly," *IEEE Transactions on Learning Technologies*, 2013, vol. 6, no. 4, pp. 389-402.

[7] C. Kenny and C. Pahl, "Intelligent and adaptive tutoring for active learning and training environments," *Interactive Learning Environments*, 2009, vol. 17, no. 2, pp. 181-195.

[8] D. Arnau, M. Arevalillo-Herraez, and J. A. Gonzalez-Calero, "Emulating human supervision in an intelligent tutoring system for arithmetical problem solving," *IEEE Transactions on Learning Technologies*, 2014, vol. 7, no. 2, pp. 155-164.

[9] B. S. Bloom, "The 2 Sigma problem: The search for methods of group instruction as effective as one-to-one tutoring," *Educational Researcher*, 1984, vol. 13, no. 6, pp. 4-16.

[10] W. L. Johnson, J. W. Rickel, and J. C. Lester, "Animated pedagogical agents: Face-to-face interaction in interactive learning environments," *International Journal of Artificial Intelligence in Education*, 2000, vol. 11, pp. 47-78.

[11] U. Ocepek, Z. Bosnić, I. N. Serbec, and J. Rugelj, "Exploring the relation between learning style models and preferred multimedia types," *Computers & Education*, 2013, vol. 69, pp. 343-355.

[12] V. Aleven, B. M. McLaren, and J. Sewall, "Scaling up programming by demonstration for intelligent tutoring systems development: An open-access web site for middle school mathematics learning," *IEEE Transactions on Learning Technologies*, 2009, vol. 2, no. 2, pp. 64-78.

[13] J. C. Stamper, T. M. Barnes, and M. Croy, "Enhancing the automatic generation of hints with expert seeding," *International Journal of Artificial Intelligence in Education*, 2011, vol. 21, no. 1-2, pp. 153-167.

[14] M. Melia and C. Pahl, "Constraint-based validation of adaptive e-learning courseware," *IEEE Transactions on Learning Technologies*, 2009, vol. 2, no. 1, pp. 37-49.

[15] A. S. Smith-Atakan and A. Blandford, "ML tutor: An application of machine learning algorithms for an adaptive web-based information system," *International Journal of Artificial Intelligence in Education*, 2003, vol. 13, pp. 235-261.

[16] J. R. Anderson *et al*., "Cognitive tutors: Lessons learned," *Journal of the Learning Sciences*, 1995, vol. 4, no. 2, pp. 167-207.

[17] K. R. Koedinger *et al*., "Intelligent tutoring goes to school in the big city," *International Journal of Artificial Intelligence in Education*, 1997, vol. 8, pp. 30-43.

[18] A. Mitrovic, "Modeling domains and students with constraint-based modeling," *Advances in Intelligent Tutoring Systems*, 2010, pp. 63-80.

[19] A. Mitrovic *et al*., "Intelligent tutor for all: The constraint-based approach," *IEEE Intelligent Systems*, 2007, vol. 22, no. 4, pp. 38-45.

[20] A. Mitrovic and S. Ohlsson, "Constraint-based knowledge representation for individualized instruction," *Computer Science and Information Systems*, 2006, vol. 3, pp. 1-22.

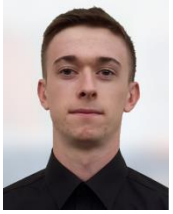
[21] P. Fournier-Viger *et al*., "A multiparadigm intelligent tutoring system for robotic arm training," *IEEE Transactions on Learning Technologies*, 2013, vol. 6, no. 4, pp. 364-377.

[22] V. Aleven *et al*., "A new paradigm for intelligent tutoring systems: Example-tracing tutors," *International Journal of Artificial Intelligence in Education*, 2009, vol. 19, no. 2, pp. 105-154.

[23] L. Razzaq *et al*., 'The ASSISTment builder: Supporting the life cycle of tutoring system content creation," *IEEE Transactions on Learning Technologies*, 2009, vol. 2, no. 2, pp. 157-166.

[24] T. J. Parr and R. W. Quong, "ANTLR — A predicated-LL(k) parser generator," *Software-Practice & Experience*, 1995, vol. 25, no. 7, pp. 789-810.

[25] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM Journal on Computing*, 1989, vol. 18, no. 6, pp. 1245-1262.



Tadej Matek received his BSc degree in 2015 from the Faculty of Computer and Information Science, University of Ljubljana. He is currently applying for a MSc in computer and information science at the same institution. His research interests include machine learning, network analysis, intelligent tutoring systems and data management systems.

Aljaž Zrnec graduated in 1999 and received the master's degree in 2002 from the Faculty of Computer and Information Science, University of Ljubljana. He received his PhD degree in 2006 from the Faculty of Computer and Information Science, University of Ljubljana in method engineering.

He works in the Laboratory for data technologies as a lecturer and assistant in the fields of databases, information system strategic planning and data integration. His research is focused on database management systems, methods for plagiarism detection and intelligent tutoring systems. He is the author or coauthor of numerous articles in professional and scientific publications.

Dejan Lavbič received his PhD degree in 2010 and is currently employed at Laboratory for Data Technologies, Faculty of Computer and Information Science, University of Ljubljana as Assistant Professor. His research interests are intelligent agents, knowledge management, social network analysis, Semantic Web, Information Quality and Intelligent tutoring systems. He is a reviewer, member of program board of several international conferences and author of more than 40 journal papers and papers presented at international conferences.