Adaptivni pristop k učenju jezika SQL

Tadej Matek, Dejan Lavbič
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana
matektadej@gmail.com; Dejan.Lavbic@fri.uni-lj.si

Izvleček

Jezik SQL je danes standard na področju poizvedovanja in manipulacije s podatki. Večina podjetij aktivno uporablja jezik SQL za podporo delovanja informacijskih sistemov, zato so potrebe po usposobljenem kadru velike. Sam proces učenja jezika SQL je težaven, saj pogosto prihaja do napačnega razumevanja temeljnih konceptov jezika. V okviru raziskave smo se lotili razvoja adaptivnega sistema za generiranje namigov kot pomoč pri procesu učenja. Sistem uporablja informacije iz zgodovinskih podatkov, iz preteklih poskusov reševanja nalog iz jezika SQL. Za odkrivanje znanja, skritega v podatkih, smo uporabili odločitvene procese Markova, ki omogočajo napovedovanje v negotovih razmerah. Poleg sistema smo razvili tudi komponento za procesiranje jezika SQL ter preprost spletni vmesnik. Preprosta evalvacija je pokazala, da je sistem zmožen generirati namige, prilagojene posameznemu študentu.

Ključne besede: inteligentni sistemi za učenje, učenje jezika SQL, odločitveni proces Markova, razčlenjevalnik jezika SQL, strojno učenje, priporočilni sistemi.

Abstract

Adaptive Approach to Learning SQL

SQL is nowadays the standard in the field of data retrieval and manipulation. The majority of companies actively use SQL as part of their business process. Subsequently, the demand for qualified personnel is high. The process of learning SQL turns out to be challenging as students often misinterpret fundamental concepts of the language. In this study, we set out to develop an adaptive system for hint generation in order to assist students in the SQL exercise-solving process. The system is based on a set of historical data and past attempts at related SQL exercises. We have employed the Markov decision processes to encode the knowledge hidden within our data as well as to make predictions under ambiguous circumstances. In addition to the system, we have also developed an SQL language parser and a simple web interface. A straightforward evaluation has shown that the system is capable of providing hints tailored to the needs of individual students.

Keywords: intelligent tutoring systems, SQL learning, Markov decision process, SQL language parser, machine learning, recommendation systems.

1 UVOD

Jezik SQL (angl. Structured Query Language) je postal standard za poizvedovanje po relacijskih podatkovnih bazah ter definiranje in manipulacijo podatkov. K splošnemu sprejetju jezika je prispevala tudi njegova prva standardizacija ANSI (angl. American National Standards Institute) leta 1986 ter ISO (angl. International Organization for Standardization) leta 1987. Večina podjetij, ki se ukvarja z razvojem programskih rešitev, danes aktivno uporablja SQL za poizvedovanje in manipulacijo s podatki, zato so potrebe po kadru s kakovostnim znanjem jezika SQL velike.

Učenje jezika SQL primarno poteka na visokošolskih izobraževalnih ustanovah, in sicer v okviru učenja podatkovnih baz ter sistemov za upravljanje s podatki. Glavni cilj učenja jezika SQL je usposobiti kandidata, da je zmožen manipulirati s podatki ter iz njih pridobiti uporabne informacije. Reševanje nalog s področja SQL poteka v okolju, podobnem profesionalnemu, da bi študent privzel način pisanja poizvedb, kakršnega bo kasneje uporabljal v delovnem okolju.

Večina poizvedb, napisanih v jeziku SQL, je preprostih in kratkih, torej bi pričakovali, da bo učenje jezika SQL hitro in učinkovito. Izkaže se, da imajo študentje mnoge težave pri učenju jezika (Prior in Lister, 2004). Prior in Lister poudarjata, da mora študent napisati poizvedbo, ki jo sistem za upravljanje s podatki pred izvedbo dodatno pretvori. Operacije, ki jih opravi sistem za upravljanje s podatki, so študentu nepoznane, tako da se težavnost znatno poveča, če študent nima takojšnje povratne informacije o rezultatu poizvedbe. Dodatna težava je tudi pomnjenje podatkovnih shem, saj študentje pozabljajo imena tabel in atributov. Napačno razumevanje do-

ločenih konceptov, kot so agregacija podatkov, omejena agregacija podatkov, združevanje tabel (angl. join), je pogosto (Mitrovic, 1998).

Današnji sistemi za učenje delno ublažijo težave študentov pri učenju. Večina jih omogoča testiranje napisanih poizvedb, tako da ima študent povratno informacijo o rezultatu podane rešitve (Prior in Lister, 2004). Takšen pristop izboljša učinkovitost učenja ter poveča motivacijo študentov za poizkušanje pisanja pravilnih rešitev. Za lažje pomnjenje shem besedilu naloge pogosto dodamo sliko dela logičnega podatkovnega modela z imeni atributov in tabel.

Kljub omenjenim izboljšavam učenje ostaja neučinkovito, saj obstajajo okoliščine, ko študent ne ve, kako bi nadaljeval. Preprost primer je scenarij, po katerem študent izbere napačen pristop k reševanju naloge. Sistem mu sicer omogoča testiranje poizvedbe, vendar je majhna verjetnost, da bo študent zmožen preiti iz napačnega pristopa reševanja k pravilnemu (zamenjati pristop k reševanju problema). Drug primer je scenarij, po katerem študent ne dojame, kaj od njega zahteva naloga. Takrat bi bilo konstruktivno študentu ponuditi idejo, kako se lotiti reševanja naloge. Omenjena primera opisujeta inteligentne sisteme za učenje, pri katerih študent dobi pomoč v obliki namigov. S pomočjo namigov je učenje do določene mere hitrejše in učinkovitejše.

V okviru raziskave smo razvili nov sistem za podporo učenja jezika SQL, ki izboljšuje učinkovitost učenja, tako da ponuja možnost personaliziranih namigov. Temelj sistema so metode umetne inteligence, ki zagotavljajo, da je generiranje namigov povsem samostojno, to pomeni, da ni potrebe po ekspertih za vnos znanja.

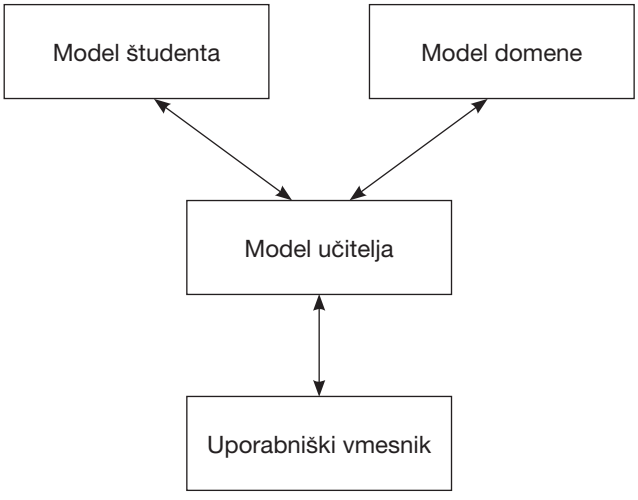
V naslednjem razdelku predstavimo trenutno stanje na področju inteligentnih sistemov za učenje. V tretjem razdelku predstavimo naš sistem z vidika arhitekture ter povezanosti posameznih komponent, v četrtem razdelku bolj podrobno predstavimo vsako izmed komponent, ki sestavljajo sistem. V petem razdelku predstavimo rezultate vrednotenja delovanja sistema. V šestem razdelku povzamemo sklepne ugotovitve in omenimo možnosti za nadaljnje delo na tem področju.

2 INTELIGENTNI SISTEMI ZA UČENJE

Inteligentni sistemi za učenje (ITS, angl. Inteligent Tutoring System) se od običajnih računalniško podprtih sistemov za učenje razlikujejo v prilagajanju uporabniškim zahtevam po načinu učenja. Medtem ko so namigi in vsebina pri običajnih sistemih statični, se pri inteligentnih sistemih za učenje vsebina za vsakega študenta določi dinamično (sistem je adaptiven). Sistem za učenje je inteligenten, če izpolnjuje tri zahteve (Polson in Richardson, 2013):

1. sistem dovolj dobro pozna domeno, v kateri deluje, da lahko samostojno rešuje probleme v tej domeni;
2. sistem je zmožen razpoznati, do kolikšne mere je študent usvojil znanje, ki ga želimo podati;
3. sistem je sposoben prilagajati težavnost nalog, da zmanjša razkorak med znanjem eksperta domene in znanjem študenta.

Glede na zgornje zahteve lahko v vsakem inteligentnem sistemu za učenje identificiramo tri ključne komponente: model domene (angl. domain model), model študenta (angl. student model) in model učitelja (angl. instructor model).

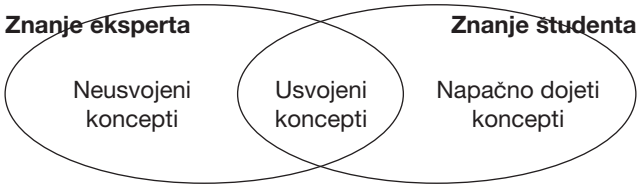


Slika 1: **Osnovna arhitektura inteligentnih sistemov za učenje**

Model domene predstavlja obsežno zbirko znanja iz domene, ki jo sistem lahko uporabi za reševanje in evalvacijo nalog. Način pridobivanja zbirke znanja je odvisen od posameznega sistema. Običajno se znanje vnese ročno s pomočjo ekspertov domene. Vneseno znanje je lahko tudi dinamično. Zbirka znanja je skupna vsem študentom, medtem ko je model študenta različen od posameznika do posameznika, odvisno od pridobljenega znanja ter količine rešenih nalog (Martin, 2002; Polson in Richardson, 2013). Model študenta lahko definiramo tudi kot okno v model domene, saj študent v določenem trenutku obvlada

podmnožico celotne domene. Običajno takšen model vsebuje informacije, kot so: katere naloge je študent rešil, katera poglavja je obiskal in, v idealnih razmerah, katere koncepte je usvojil in katerih ne.

Slika 1 prikazuje, kako omenjene komponente med seboj sodelujejo v procesu učenja. Model učitelja uporablja informacije o posameznem študentu, ki jih pridobi iz modela študenta (adaptivni del sistema). Skupaj z informacijo o domeni oblikuje namige ter izbira prihajajoče naloge. Pomemben dodatek v nekaterih sistemih predstavlja model odstopanj (angl. perturbation model) (Martin, 2002).



Slika 2: **Model odstopanj v inteligentnih sistemih za učenje z vidika študenta**

Kot je razvidno s slike 2, lahko študent določene koncepte dojame napačno. Zato mnogi sistemi vsebujejo modele za določanje, katere koncepte mora študent usvojiti ponovno. Model odstopanj je običajno predstavljen kot zbirka pogostih napak, h katerim so študentje nagnjeni med reševanjem. Tako lahko sistem zazna študentove napake.

Skozi razvoj računalniško podprtega učenja se je razvilo več inteligentnih sistemov za učenje. Najbolj uveljavljeni med njimi so kognitivni tutorji (angl. cognitive tutors), novejši pristopi uporabljajo koncept omejitev ali zgradijo model študenta s pomočjo strojnega učenja. V nadaljevanju so opisani omenjeni pristopi k izgradnji inteligentnih sistemov za učenje.

2.1 Kognitivni tutorji

Dolga leta so bili kognitivni tutorji najaktualnejša rešitev na področju inteligentnih sistemov za učenje. Čeprav so danes v razvoju nove metode, so kognitivni tutorji še vedno pomemben gradnik v računalniško podprtem učenju.

Leta 1982 je bila dokončana teorija ACT* (angl. Adaptive control of thought), na podlagi katere je nastala večina današnjih inteligentnih sistemov za učenje (Anderson, Corbett, Koedinger, in Pelletier, 1995). Glavni prispevek omenjene teorije je, da lahko procese človeškega mišljenja delimo na deklarativne in proceduralne. Razlaga za delitev je dokaj preprosta. Če želimo opraviti določeno nalogo, potrebujemo določena proceduralna znanja. Človek mora, preden pridobi ustrezna proceduralna znanja, usvojiti ustrezno deklarativno znanje. Bistveno za učenje je, da deklarativno znanje usvojimo vsaj enkrat. Tudi če kasneje ne poznamo potrebnega deklarativnega znanja, lahko še vedno opravimo nalogo, če smo le usvojili ustrezno proceduralno znanje.

Za doseganje kompetence v določeni domeni bi bilo glede na teorijo dovolj, da usvojimo celotno deklarativno znanje, vendar bi interpretacija deklarativnega znanja brez proceduralnih pravil povzročila preveliko obremenitev delovnega spomina posameznika. Velja tudi nasprotno – za doseganje kompetence v domeni bi lahko celotno znanje usvojili v proceduralni obliki (Martin, 2002). Zaradi prevelikega števila proceduralnih pravil tudi takšna alternativa ni sprejemljiva. Torej za uspešno učenje potrebujemo zadostno količino tako deklarativnega kot tudi proceduralnega znanja. Pridobivanje deklarativnega znanja ni težavno, saj nam ga podajajo učitelji v obliki definicij in izrekov. Večja težava je pridobitev proceduralnega znanja, saj mora vsak posameznik skozi reševanje nalog usvojiti ustrezne postopke. Poudarek kognitivnih tutorjev je zato predvsem na pridobivanju proceduralnih znanj.

Kognitivni tutorji so bili sprva razviti v namene potrjevanja teorije ACT*. Zaradi osredotočenosti na proceduralno znanje predpostavljajo, da študentje že imajo potrebno deklarativno znanje. Njihov model domene predstavlja nabor proceduralnih pravil. Cilj učenja je spodbuditi študenta, da se obnaša, kot določajo proceduralna pravila v modelu domene. Učenje poteka po metodi sledenja modelu (angl. model-tracing) (Martin, 2002). Med reševanjem študent oddaja nepopolne rešitve. Sistem sledi študentu med reševanjem in mu v primerih, ko zaide s pravilne poti, ponudi namig. Če sistem ne prepozna študentove akcije, ga obvesti o splošni napaki. Zaradi kombinatorične zahtevnosti sledenja študentovi rešitvi skozi celoten model domene je študent velikokrat prisiljen, da se vrne na pravilno pot reševanja. Za izgradnjo modela študenta se uporabi Bayesova verjetnost. Izračuna se verjetnost, da je študent usvojil proceduralno pravilo. Izračun se izvede vsakič, ko sistem ugotovi, ali je rešitev pravilna ali napačna.

Eno izmed prvih orodij na področju kognitivnih tutorjev je bilo orodje LISP tutor za učenje istoimen-

skega programskega jezika. Tutor je deloval tako, da je študentu ponudil predlogo programske kode, ki jo je bilo treba dopolniti. V primeru, da je študent zašel s pravilne poti, je vskočil program in zamenjal napačno funkcijo s pravilno. Orodje se je izkazalo za zelo uspešno in je bilo preizkušeno v univerzitetnem okolju.

2.2 Modeliranje na podlagi omejitev

Največja težava kognitivnih tutorjev je njihova omejenost. Kognitivni tutorji usmerjajo študenta po natančno določeni poti, ki je bila predvidena ob vnosu proceduralnih pravil. Študent tako postane odvisen od pomoči tutorja, ki ga vodi do rešitve. Za proceduralne domene, kot je npr. aritmetika, je takšno delovanje povsem ustrezno. V kompleksnejših, deklarativnih domenah, kot je jezik SQL, postane učenje težavno, saj študent ne poskuša sam reševati naloge, temveč se zanaša na orodje, da ga vodi k rešitvi. Jedro težave pomeni dejstvo, da je metoda sledenja modelu preveč restriktivna, saj ne upošteva, da imajo lahko naloge več rešitev. Neupoštevanje več rešitev je posledica vnašanja proceduralnih pravil, saj je za vsako nalogo vnesena le ena pravilna pot do rešitve. Težave se ne da preprosto odpraviti tako, da bi vnesli več proceduralnih pravil za določeno nalogo, saj obstaja preveč kombinacij.

Kot odgovor na omenjene pomanjkljivosti so razvili nov pristop k modeliranju inteligentnih sistemov za učenje, katerega temelj so omejitve (angl. constraints) (Mitrovic, 2010; Mitrovic, Martin in Suraweera, 2007; Mitrovic in Ohlsson, 2006; Mitrovic, Ohlsson in Barrow, 2013). Leta 1994 je Ohlsson pripravil novo teorijo človeškega učenja, ki se močno razlikuje od obstoječe teorije ACT*. Za razliko od teorije ACT* je Ohlsson trdil, da se naučimo proceduralnih pravil, ko ugotovimo, da smo med reševanjem napravili napako. Ohlssonova teorija trdi tudi, da je pojav napak med reševanjem pogost in običajen pojav, saj je naš delovni spomin preobremenjen. V danem trenutku sicer imamo potrebno deklarativno znanje, vendar obstaja preveliko število kombinacij, da bi se lahko odločili pravilno. Ko enkrat usvojimo potrebno proceduralno znanje, se lažje odločamo, katere dele deklarativnega znanja je treba uporabiti.

Ohlsson uporablja omejitve za opisovanje preslikav med delčki deklarativnega znanja in trenutnimi razmerami. Vsaka omejitev je sestavljena iz dveh komponent: iz pogoja relevantnosti, ki pove, ali je delček deklarativnega znanja relevanten, ter iz pogoja zadoščenosti, ki podaja, ali je bil – v primeru, da je pogoj relevantnosti izpolnjen – delček deklarativnega znanja uporabljen pravilno.

Model domene je tako predstavljen kot zbirka omejitev, ki jim mora študent zadostiti. V določenem trenutku se celoten nabor omejitev preveri nad rešitvijo študenta. Če je določena omejitev relevantna (pogoj relevantnosti je izpolnjen), se preveri pogoj zadoščenosti. V primeru, da je tudi pogoj zadoščenosti izpolnjen, lahko predpostavimo, da je študent usvojil koncept, ki ga modelira omejitev. V nasprotnem primeru študent ni izpolnil omejitve, omejitev postane del modela odstopanj. Sistem lahko uporabi zbirko neizpolnjenih omejitev, da poskuša ponovno priučiti študenta konceptov, ki jih ni dojel. Model študenta ravno tako temelji na omejitvah, in sicer zajema omejitve, ki jih je študent usvojil.

V sklopu raziskave (Mitrovic idr., 2013) je bilo razvito orodje SQL-Tutor, ki uporablja modeliranje z omejitvami za učenje jezika SQL. Omejitve v sistemu bodisi preverjajo rešitev študenta z idealno rešitvijo bodisi preverjajo sintaktično pravilnost rešitve. Omejitve so zapisane v programskem jeziku LISP. Za preverjanje pogoja zadoščenosti in pogoja relevantnosti se uporablja ujemanje vzorcev (angl. pattern-matching). Za potrebe delovanja orodja je bila pripravljena obsežna zbirka omejitev. Orodje se je izkazalo za dokaj uspešno. Raziskava navaja, da se znanje študentov izboljša že po dveh urah uporabe.

2.3 Metode s področja umetne inteligence

Razvoj področja umetne inteligence je prinesel s seboj tudi pojav novih področij, kot sta strojno učenje in rudarjenje podatkov. Omenjeni metodi sta zelo obetajoči in uporabni tudi v inteligentnih sistemih za učenje. Gradnja inteligentnih sistemov za učenje je namreč časovno in stroškovno potratna operacija. Razlog se nahaja v naravi določanja modela domene pri kognitivnih tutorjih in sistemih, ki temeljijo na omejitvah. Da inteligentni sistem za učenje doseže določeno mero uporabnosti, je treba vložiti več sto ur vnašanja pravil in omejitev, ki predstavljajo zbirko znanja (model domene). Metode iz umetne inteligence so nam v pomoč, saj do določene mere omogočijo avtomatsko generiranje zbirke znanja iz zunanjih podatkov. Seveda to velja le pod predpostavko, da imamo za posamezno domeno na voljo dovolj podatkov, iz katerih se lahko učimo.

Eden izmed prvih poskusov avtomatizacije generiranja modela domene je dokumentiran v delu Jarvis, Nuzzo-Jones in Heffernan (2004). Temeljni cilj razvitega sistema je omogočiti ekspertom domen, neveščim programiranja, da izdelajo vsebino za učenje v inteligentnih sistemih za učenje. Eksperti domen bi vnašali pravila – programiranje po zgledu (angl. programming by demonstration) –, pri čemer bi sistem iz opazovanja učiteljev med reševanjem sam zgradil ustrezne programske konstrukte. Avtorji sistema so uporabili strojno učenje za avtomatizacijo generiranja produkcijskih pravil, ki se uporabljajo pri učenju in podajanju namigov. Sistem se je izkazal kot zelo uspešen. Testiranje je bilo opravljeno na primerih učenja seštevanja ulomkov, večstolpčnega seštevanja ter igre tri v vrsto. Sistem je zmožen za skoraj vse primere izdelati natančen predpis proceduralnih pravil, ki se lahko uporabijo v procesu učenja. Poleg avtomatičnega generiranja modela domene se je razvila še vrsta drugih pristopov na podlagi umetne inteligence (Aleven, McLaren in Sewall, 2009; Fournier-Viger, Nkambou, Nguifo, Mayers in Faghihi, 2013; Stamper, Barnes in Croy, 2011).

Vse večji pomen imajo tudi zgodovinski podatki, ki jih lahko beležimo med potekom učenja. Dovolj velika količina podatkov oziroma dovolj kakovostni podatki nam omogočajo, da iz njih izluščimo koristne informacije. Tako lahko iz podatkov o učenju razberemo, pri čem študentje napravijo največ napak, katere naloge rešijo zadovoljivo itd. Prednost zgodovinskih podatkov so začeli izkoriščati tudi inteligentni sistemi za učenje. Iz shranjenih rešitev študentov je mogoče zgraditi bazo znanja, ki se lahko uporabi kot model domene. Eden izmed takšnih sistemov je t. i. Hint factory (Barnes, Stamper, Lehman in Croy, 2008), ki za svoje delovanje uporablja odločitvene procese Markova. Omenjeni sistem je podlaga našega sistema in je bolj podrobno opisan v nadaljevanju.

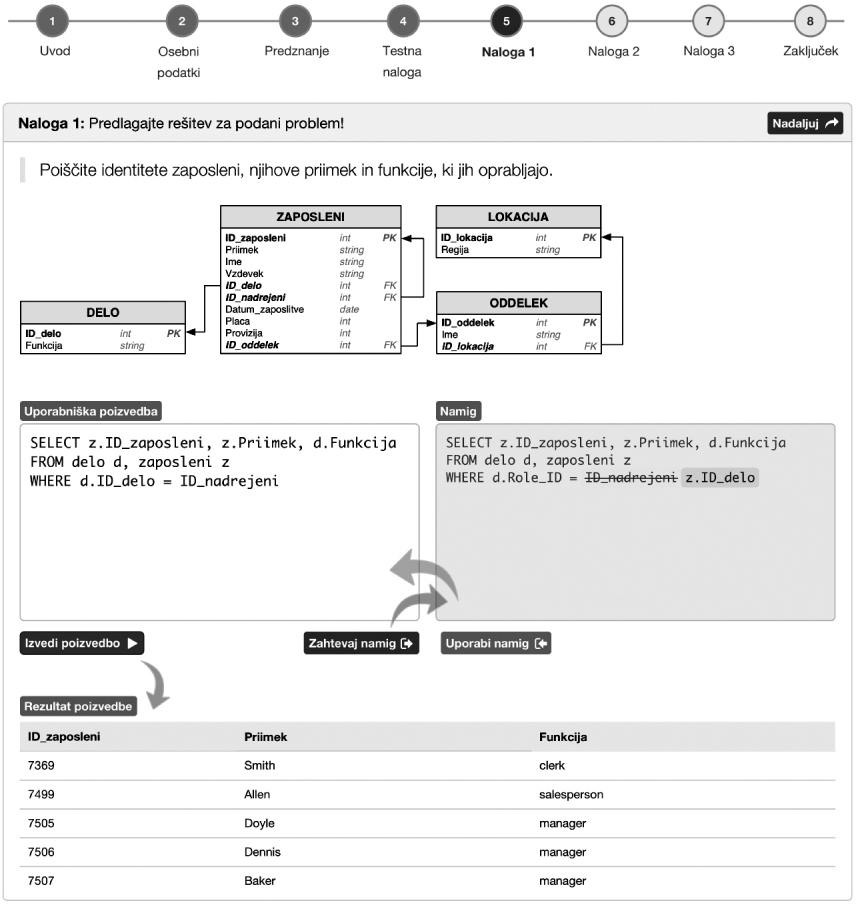
3 PRIPOROČILNI SISTEM

Po pregledu trenutnega stanja na področju učenja jezika SQL smo ugotovili, da je večina sistemov, katerih namen je lajšanje učenja jezika SQL (kot je npr. SQL-Tutor), statičnih in neprilagodljivih. Razlog je v pristopu, ki jih takšni sistemi uporabljajo za učenje. Tako kognitivni tutorji kot modeliranje z omejitvami uporabljajo statično bazo znanja, za kar potrebujemo vrsto strokovnjakov domen, ki ročno vnašajo pravila. Več kot ima naloga različnih načinov reševanja, več pravil je potrebnih, več časa je treba vložiti za vnos teh pravil in hkrati se povečajo stroški izdelave takšnega sistema. Namigi, ki jih statični sistemi ponudijo študentu, so splošni in neprilagojeni stanju, v katerem se nahaja študent. Kolikor nam je znano, še ne obstaja sistem, ki bi uporabljal umetno inteligenco za učenje jezika SQL.

V okviru raziskave je bil izdelan sistem za lajšanje učenja jezika SQL (Matek, 2015). Glavna motivacija pri izdelavi inteligentnega sistema so bili zgodovinski podatki reševanja nalog iz domene jezika SQL. Na voljo so nam bili podatki reševanja nalog iz let 2014 in 2015 v okviru obveznega predmeta Osnove podatkovnih baz. Predmet je v programu prvega letnika dodiplomskega študija na Fakulteti za računalništvo in informatiko in pomeni uvod v sisteme za upravljanje s podatki. Podatki so med drugim vsebovali čas oddaje rešitve, identifikator naloge in sheme ter dejansko poizvedbo, ki predstavlja rešitev študenta. Vseh zabeleženih rešitev je bilo približno 32.000. Znanje, skrito v naših podatkih, smo želeli uporabiti za izdelavo sistema, ki bo v pomoč študentom pri reševanju tovrstnih nalog. Prednosti sistema so v priporočilnem modulu in njegovem generiranju namigov. Poleg namigov sistem omogoča tudi druge, že uveljavljene prednosti, kot so testiranje poizvedb med reševanjem ter prilaganje delov slik logičnega podatkovnega modela za lažjo vizualizacijo.

Izbrali smo metode umetne inteligence za izgradnjo sistema. Odločili smo se za uporabo strojnega učenja nad podatki ter napovedovanje naslednjega koraka iz trenutnega stanja študentove rešitve. Natančneje, naš sistem temelji na metodi Hint factory (Barnes idr., 2008), ki uporablja odločitvene procese Markova (MDP, angl. Markov Decision Process) za izgradnjo modela domene ter vrednostno iteracijo (angl. value iteration) za določanje ocen stanj. Omenjena metoda do sedaj še ni bila uporabljena v domeni učenja jezika SQL, zato je zanimivo opazovati učinkovitost metode v tej domeni. Odločitev o uporabi omenjene metode temelji na uporabi zgodovinskih podatkov in tudi na sami naravi metode. Metoda je namreč prilagojena za učenje iz zgodovinskih podatkov ter napovedovanje naslednjih korakov reševanja iz obstoječih podatkov. Poleg strojnega učenja je pomembnejša komponenta sistema tudi komponenta za analizo in procesiranje samega jezika SQL, ki je sestavni del priporočilnega modula

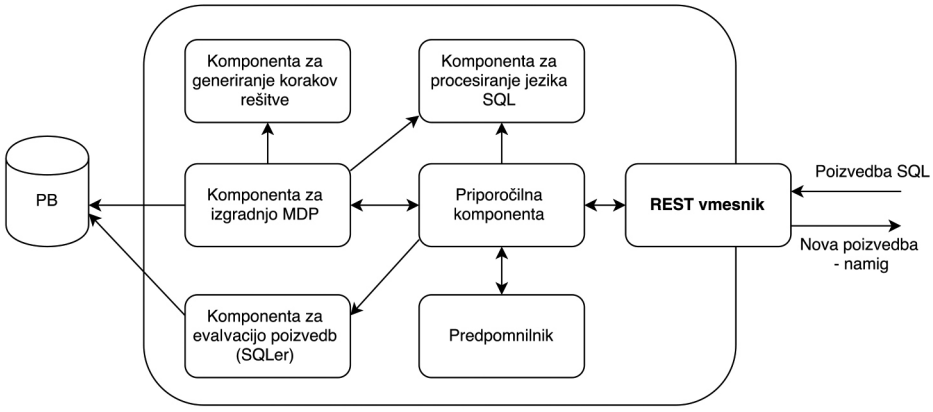
na sliki 4. V procesu preslikave študentove rešitve na model domene je pomembna primerjava dveh poizvedb SQL, zato je takšna komponenta nujno potrebna. Dodatna zahteva pri zasnovi sistema je bila, da namigi ne razkrijejo celotne rešitve, temveč ponudijo študentu le prihodnje stanje – naslednji korak na poti do rešitve. Tako postanejo namigi dejansko uporabni.



Slika 3: **Primer zaslonske maske za reševanje nalog**

Poleg priporočilnega modula sistem vsebuje tudi modul za upravljanje z nalogami. Učitelji in strokovnjaki domene lahko preprosto prek spletne aplikacije vnašajo nove naloge, urejajo obstoječe rešitve ter določajo idealne, ki pomagajo v procesu generiranja namigov itd. Spletna aplikacija poleg upravljanja nalog omogoča tudi simulacijo reševanja nalog ter prikaz namigov. Primer zaslonske maske spletne aplikacije je viden na sliki 3, in sicer za primer simulacije reševanja naloge. Kot je razvidno iz zaslonske maske, lahko študent med reševanjem zahteva namig (gumb »Zahtevaj namig«) in testira poizvedbo (gumb »Izvedi poizvedbo«). Vrnjeni namig lahko uporabi (gumb »Uporabi namig«), pri čemer se poizvedba študenta zamenja s poizvedbo iz namiga (poizvedba na desni, kjer so označene spremembe v primerjavi s študentovo poizvedbo).

Slika 4 prikazuje arhitekturo priporočilnega dela zalednega sistema. Proces generiranja namigov poteka tako, da komunikacija spletne aplikacije z zalednim delom sistema poteka preko spletnih storitev, in sicer spletne storitve REST. Kot vhod sistem prejme poizvedbo SQL, za katero študent želi namig oziroma dopolnitev. Sama spletna storitev v nadaljnjih korakih komunicira s priporočilnim modulom. Priporočilni modul skrbi za posodobitve modela domene ter za samo generiranje namigov. Model domene je – glede na metodo Hint factory – predstavljen kot kombinirana množica zgodovine rešitev, predstavljenih z grafom MDP. Za kreiranje objekta



Slika 4: **Shema arhitekture priporočilnega modula**

MDP iz zgodovinskih podatkov priporočilni modul uporablja komponento za izgradnjo MDP. Ta skrbi za več procesov. Sprva pretvori poizvedbo SQL iz zgodovinskih podatkov v drevesno strukturo s pomočjo komponente za procesiranje jezika SQL. Nad dobljeno drevesno strukturo nato opravi še dodatno transformacijo s pomočjo komponente za generiranje korakov rešitve. Rezultat transformacije je množica dreves (gozd), ki si sledijo v sosledju in predstavljajo korake na poti reševanja določene poizvedbe SQL. Komponenta nato združi vse korake reševanja v objekt MDP in postopek ponovi za vse zgodovinske zapise v podatkovni bazi. Kot rezultat dobimo velik skupni objekt MDP, ki vsebuje vse poti reševanja, ki so jih izbrali študentje pri reševanju nalog. Nastali MDP se vrne priporočilni komponenti, ta pa ga vstavi v predpomnilnik. Ob naslednjem dostopu je tako MDP predpomnjen in ga lahko priporočilna komponenta pridobi neposredno brez uporabe drugih komponent. Dodatna prednost predpomnjenja je ta, da se izognemo časovno potratnim operacijam nad podatkovno bazo. Priporočilna komponenta po pridobitvi objekta MDP izvede ujemanje najboljšega stanja glede na ocene stanj in trenutno stanje, v katerem se nahaja študent. Kot namig se vrne poizvedba v iskanem stanju.

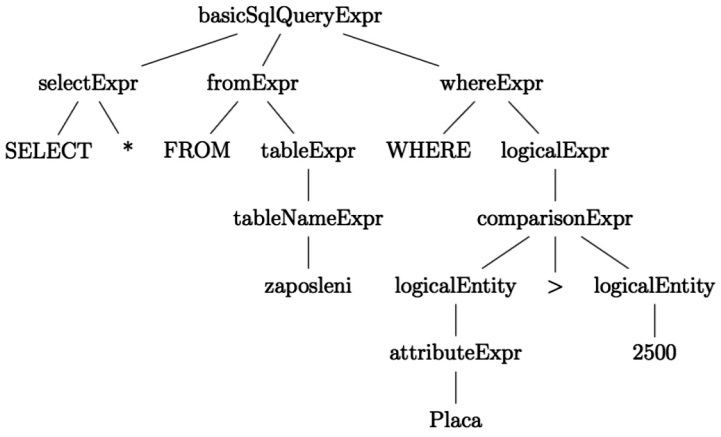
Ker se sistem neprestano prilagaja in dopolnjuje svojo zbirko znanja, priporočilna komponenta dostopa tudi do komponente za evalvacijo poizvedb. Tako je sistem zmožen za vsako vhodno poizvedbo določiti oceno ter jo vstaviti v podatkovno bazo kot dodaten zgodovinski zapis. Ker je objekt MDP nespremenljiv (angl. immutable), bo dodani poskus upoštevan ob naslednji ponovni gradnji MDP. Sistem zato ob določenih časovnih intervalih izprazni predpomnilnik, da zagotovi ponovno gradnjo MDP. Omeniti je treba, da je sistem zmožen ponuditi namig le, če je vhodna poizvedba sintaktično pravilna ter se pravilno izvede.

4 PODROBEN OPIS KOMPONENT

V prejšnjem razdelku je sistem opisan z vidika medsebojnega delovanja komponent. Ta razdelek je namenjen podrobnemu opisu vsake izmed komponent priporočilnega modula zaledenga sistema.

4.1 Komponenta za procesiranje jezika SQL

Za potrebe procesa analize poizvedb SQL je nujno potrebna komponenta, ki je zmožna pretvoriti poljubno pravilno obliko poizvedbe SQL v strukturo, ki je bolj primerna za procesiranje z vidika računalniškega sistema. Komponenta za procesiranje jezika SQL skrbi za ustrezno pretvorbo poizvedb SQL v drevesno strukturo, ki se shrani v pomnilniku. Za svoje delovanje komponenta uporablja razčlenjevalnik jezikov ANTLR (Parr in Quong, 1995), pri čemer je bil v okviru raziskave izdelan slovar za jezik SQL. Poleg orodja ANTLR bi za procesiranje jezika lahko uporabili regularne izraze, a regularni jeziki nimajo dovolj velike izrazne moči za procesiranje kompleksnih jezikov, kot je SQL. Primer drevesne strukture po razčlenjevanju poizvedbe je na sliki 5.



Slika 5: **Primer drevesne strukture za preprosto poizvedbo SQL**

4.2 Komponenta za generiranje korakov rešitve

Komponenta za procesiranje jezika SQL in zgodovinski podatki sami po sebi ne predstavljajo zadostnega pogoja za uspešno generiranje namigov. Če želimo zadostiti zahtevi, naj namig ne razkrije celotne rešitve, temveč le naslednji korak na poti do končne rešitve, potrebujemo posamezne korake reševanja. Koraki reševanja nam iz zgodovinskih podatkov niso na voljo, saj je zabeležena le celotna poizvedba SQL. Za podporo korakov reševanja bi morali sproti, med študentovim reševanjem naloge, na strežnik pošiljati delne rešitve. Izkaže se, da lahko ob predpostavki, da študent rešuje nalogo v vrstnem redu glede na sklope, korake rešitve ustvarimo iz obstoječih poizvedb.

Komponenta za generiranje korakov rešitve skrbi za ustvarjanje korakov reševanja iz shranjenih poizvedb SQL. Temelji na prej omenjeni predpostavki reševanja po sklopih. Kot vhod komponenta prejme drevesno strukturo, ki predstavlja poizvedbo SQL. Kot rezultat vrne gozd, ki predstavlja množico poizvedb, posamezna poizvedba predstavlja korak reševanja. Ker smo predpostavili, da reševanje poteka po sklopih, lahko privzamemo, da se študent pomika po drevesni strukturi od leve proti desni. Sprva napiše sklop SELECT, ki je v drevesni strukturi najbolj levo, kot zadnjega napiše morebiten sklop LIMIT, ki je najbolj desno. Korake reševanja lahko zato dobimo z obhodom v globino (angl. depth-first) po drevesni strukturi, ki predstavlja poizvedbo. Komponenta izvede omenjeni obhod drevesa, pri čemer se ustavi le pri listih drevesa. Listi drevesa namreč predstavljajo uporabnikov vnos in ne sintaktičnega pravila. Ko komponenta doseže list drevesa, naredi kopijo drevesa, ki vsebuje vsa obiskana vozlišča na poti do trenutnega lista drevesa, vključno z listom. Omenjeni pristop ni najbolj ekonomičen, saj za vse liste drevesa naredi kopijo drevesa in jih doda v gozd. Število rešitev lahko zmanjšamo tako, da ustvarimo kopijo drevesa le v primeru, da smo v listu, ki je najbolj desni otrok očeta. Takšen pristop ne poslabša generiranja namigov, saj bi kopije drevesa pri vseh listih povzročile, da namigi vsebujejo premalo informacije (spremembe nad poizvedbo bi bile premajhne).

Poleg preverjanja pogoja najbolj desnega lista je pred vstavljanjem v gozd treba preveriti, ali kopija drevesa predstavlja sintaktično pravilno poizvedbo SQL. Pravilnost koraka rešitve preverimo z obstoječo komponento za procesiranje jezika SQL. Za drevesno strukturo na sliki 5 bi komponenta ustvarila poizvedbe, ki predstavljajo korake reševanja:

```
SELECT *

SELECT *
FROM zaposleni

SELECT *
FROM zaposleni
WHERE Placa

SELECT *
FROM zaposleni
WHERE Placa > 2500
```

Opazimo, da so koraki rešitve predpone končne poizvedbe, končnega koraka.

4.3 Komponenta za izgradnjo MDP

Sistem ima do te stopnje na voljo vse podatke, ki so potrebni za učenje in generiranje namigov. Poleg

shranjene zgodovine reševanja nalog ima na voljo tudi komponento, ki je zmožna analize jezika SQL, ter komponento, ki nam vrača korake reševanja. Manjka le še struktura ali algoritem, ki bo povezal pridobljeno znanje in se iz podatkov učil ter napovedoval naslednje pravilno stanje. Iskana struktura je MDP, ki omogoča napovedovanje v negotovih razmerah.

MDP je definiran kot peterka

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

pri čemer je končna množica stanj, končna množica akcij, ki povezuje stanja, matrika prehodnih verjetnosti, funkcija nagrad in diskontni faktor. Obnašanje sistema je nedeterministično tj. matrika prehodnih verjetnosti za vsako akcijo določa verjetnost, da nas bo akcija dejansko pripeljala v ciljno stanje. Akcije imajo lahko več ciljnih stanj, vsako ciljno stanje je dosegljivo z določeno verjetnostjo. Funkcija nagrad za vsako stanje podaja nagrado, ki jo agent prejme, če doseže to stanje. Nagradna lahko deluje tudi kot kazen, če je nagrada negativna. Politika agenta podaja preslikavo med stanji in akcijami. Vsaka politika enolično določa obnašanje agenta v sistemu, saj podaja, katero akcijo naj agent izbere v odvisnosti od stanja, v katerem se nahaja. Glede na zgornje lastnosti lahko definiramo vrednostno funkcijo:

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

Cilj MDP je najti optimalno politiko, takšno, ki maksimira vrednostno funkcijo oziroma nagrado, ki jo agent prejme v prihodnosti:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s)$$

Za določanje optimalne politike uporabimo vrednostno iteracijo, ki iterativno računa boljše ocene vrednostne funkcije za vsako stanje, dokler ocene ne konvergirajo:

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

Diskontni faktor, ki daje prednost bodisi dolgoročnim bodisi kratkoročnim rešitvam, je bil nastavljen na 1.0, kar pomeni, da preferiramo dolgoročne rešitve.

Uporaba struktur MDP je primerna za učenje jezika SQL. Ker imamo na voljo množico zgodovinskih podatkov, iz katerih lahko pridobimo korake rešitev, lahko korake združimo v samostojen odločitveni proces (pri tem pazimo, da ni podvojenih stanj) in tako dobimo enotno zbirko znanja za posamezno nalogo (zbirka znanja, ki vsebuje vse različne poti reševanja). Nato poiščemo optimalno politiko, kateri bo priporočilni modul (agent) sledil. Odločitveni proces Markova je predstavljen kot usmerjeni graf, v katerem vozlišča predstavljajo stanja, akcije pa povezave med vozlišči. Vsako vozlišče je objekt, ki vsebuje enega izmed korakov rešitev. Poleg poizvedbe vozlišče vsebuje tudi informacijo o nagradi stanja ter seznam izhodnih ter vhodnih povezav. Povezava je predstavljena z verjetnostjo, izvornim ter ciljnim stanjem. Verjetnost povezave je enaka frekvenci uporabe te povezave izmed vseh povezav, ki izhajajo iz izvornega stanja povezave. Frekvence uporabe povezav izračunamo iz zgodovinskih podatkov. Nagrado stanj lahko izračunamo s pomočjo komponente za evalvacijo poizvedb. Tako izpolnjujemo vse potrebne pogoje za uporabo vrednostne iteracije, ki določi pričakovane nagrade za vsako stanje glede na akcije stanja. Agent lahko uporabi izračunane nagrade za odločanje, katero bo naslednje stanje – naslednji korak reševanja. Odločanje poteka v priporočilni komponenti.

V primerih, ko ni dovolj zgodovinskih podatkov ali je njihova kakovost vprašljiva, lahko skrbniki sistema vnesejo eno ali več pravilnih (idealnih) rešitev za posamezno nalogo. Idealne rešitve se tako, poleg ostalih zgodovinskih podatkov, dodajo v MDP med grajenjem le-tega. Idealne rešitve si lahko predstavljamo kot seme, s katerim pospešimo generiranje namigov. Velikokrat se zgodi, da uvedemo nove naloge, za katere še ne obstajajo podatki o reševanju študentov. Idealne rešitve v omenjenih primerih izboljšajo uporabnost sistema.

4.4 Priporočilna komponenta

Priporočilna komponenta uporablja zgrajen MDP in deluje kot agent pri učenju. Kot vhod prejme poizvedbo SQL, za katero študent želi namig. Poizvedbo s pomočjo komponente za procesiranje jezika SQL pretvori v drevesno strukturo. MDP se pridobi

bodisi iz predpomnilnika bodisi prek komponente za gradnjo MDP.

Ko ima komponenta v lasti objekt MDP, najprej izvede ujemanje najbližjega stanja glede na stanje študenta. Ujemanje poteka tako, da komponenta izvede Zhangov-Shashev algoritem za primerjanje drevesnih struktur (Zhang in Shasha, 1989) nad vsakim stanjem. Na koncu izbere stanje, ki je kar najbolj podobno stanju študenta, oziroma stanje, katerega drevesna struktura je najbolj enaka drevesni strukturi poizvedbe študenta. Če najde ujemajoče stanje, pregleda sosednja stanja.

Pregled sosednjih stanj temelji na iskanju stanja z višjo oceno od trenutnega stanja. Če takšnega stanja ni, ne moremo ponuditi uporabnega namiga, saj je študent v popolnoma napačni veji reševanja ali pa preprosto nimamo dovolj zgodovinskih podatkov. V takšnih primerih zato kot namig ponudimo poizvedbo enega izmed začetnih stanj z najvišjo oceno. Če obstaja stanje z višjo oceno, ločimo dva primera. Prvi primer predstavlja stanje z višjo oceno, do katerega vodi povratna povezava. Povratna povezava nakazuje, da se študent nahaja v napačni veji in da ga lahko usmerimo nazaj na eno izmed pravilnih poti reševanja. Za usmerjanje nazaj na pravilno pot ni dovolj ponuditi namig za naslednje stanje z višjo oceno, saj je takšno stanje še vedno del napačne veje reševanja. Namesto tega se moramo vrniti do prvega skupnega prednika napačne veje in pravilne veje. Ko najdemo skupnega prednika napačne in pravilne veje reševanja, pregledamo njegova sosednja stanja. Kot namig ponudimo sosednje stanje z najvišjo oceno. Drugi primer predstavlja stanje z višjo oceno, do katerega vodi običajna povezava. Takrat preprosto ponudimo namig za naslednje stanje. V primeru več stanj z višjo ocen izberemo tistega z najvišjo. Če ne najdemo ujemajočega stanja, ponudimo namig za najbližje stanje, tj. stanje, ki je najbolj podobno stanju študenta.

4.5 Komponenta za evalvacijo poizvedb

Kot komponento za evalvacijo poizvedb se uporablja obstoječe orodje SQLer, razvito v okviru predmeta Osnove podatkovnih baz za potrebe vrednotenja poizvedb. Orodje za delovanje potrebuje delujočo povezavo s podatkovno bazo, v kateri so shranjene sheme, nad katerimi se preverjajo naloge. Orodje deluje na podlagi primerjave rezultatov poizvedbe študenta ter rezultatov idealne poizvedbe (rešitve naloge). Pri ocenjevanju se med drugim upoštevajo odvečni atributi, odvečne vrstice, neustrezen vrstni red vrstic itd.

5 EVALVACIJA SISTEMA

Evalvacija sistema je potekala na podlagi študije primerov. Primeri so bili izbrani glede na težave, s katerimi se srečujejo študentje, opisane v prvem razdelku. Testiranje je potekalo z uporabo obstoječih zgodovinskih podatkov o reševanju nalog, saj nismo imeli na voljo možnosti testiranja v učilnici. Testni podatki služijo le kot preprost vpogled v delovanje sistema. Študija primerov je potekala za dva scenarija:

- študent se loti reševanja naloge na popolnoma napačen način;
- študent delno reši nalogo in želi namig za nadaljnje reševanje.

Za vsak omenjeni scenarij so bile izbrane reprezentativne naloge s težavnostnimi stopnjami od najlaže do najtežje naloge. Poleg izbire naloge so bili izbrani tudi ustrezni zgodovinski podatki, ki so služili kot študentov vnos k nalogam.

5.1 Scenarij 1: napačen pristop k reševanju

V okviru prvega scenarija smo preverjali, do kolikšne mere je sistem sposoben ponuditi namig, če študent ubere popolnoma napačen pristop k reševanju naloge. Rezultati testiranja so podani v tabeli 1.

Tabela 1: **Študija primerov za scenarij 1**

Idealna rešitev	Rešitev študenta	Prvi in drugi namig
SELECT COUNT (*) FROM zaposleni, oddelek WHERE zaposleni.ID_oddelek = oddelek.ID_oddelek AND oddelek.Ime = 'SALES';	SELECT * FROM ODDELEK;	SELECT COUNT (*) FROM zaposleni WHERE ID_oddelek SELECT COUNT (*) FROM zaposleni WHERE ID_oddelek IN (SELECT ID_oddelek)
SELECT * FROM Zaposleni WHERE ID_oddelek IN (SELECT ID_oddelek FROM Zaposleni GROUP BY ID_oddelek HAVING COUNT(ID_oddelek) > 3 AND ID_oddelek = 30);	SELECT z1.id_zaposleni, z1.priimek, z1. ime, z1.vzdevek, COUNT(z1.ID_zaposleni) FROM zaposleni z1, zaposleni z2 WHERE z1.ID_oddelek = 30	SELECT * FROM Zaposleni WHERE ID_oddelek SELECT * FROM zaposleni WHERE ID_oddelek = 30
SELECT * FROM Zaposleni WHERE Placa > (SELECT MAX(Placa) FROM Zaposleni, Delo WHERE Zaposleni.ID_delo = Delo.ID_delo AND Funkcija = 'Salesperson');	SELECT MAX(z.Placa) FROM ZAPOSLENI z, DELO d WHERE z.ID_Delo = d.ID_Delo AND d.Funkcija = 'SALESPERSON'	SELECT * FROM ZAPOSLENI z1 WHERE Placa SELECT * FROM ZAPOSLENI z1 WHERE Placa > ALL (SELECT z2.Placa)

Opazimo, da sistem ponuja namige, ki obsegajo začetne dele poizvedb. Razlog je v tem, da se študent nahaja v popolnoma napačni veji, iz katere ga ne moremo preusmeriti nazaj do pravilne veje. Zato mu, kot je že bilo omenjeno, ponudimo namig za eno izmed začetnih stanj. Dolžina, do katere preiščemo začetna stanja za iskanje najbolj optimalnega stanja, je eden izmed parametrov sistema. V sklopu testiranja je bil parameter nastavljen na 2, kar pomeni, da preiščemo začetno stanje in njegove naslednike. Iz prve vrstice tabele 1 je razvidno, da sistem študenta z namigi ne usmerja po poti idealne rešitve, temveč po poti rešitve drugega študenta, kar je zaželeno, saj tako študent preizkusi tudi nove načine reševanja.

5.2 Scenarij 2: dopolnjevanje rešitve z namigi

Scenarij 2 predstavlja položaj, ko študent delno reši nalogo, nato pa ne zna nadaljevati z reševanjem. Razmere narekujejo generiranje namiga, ki študentu pomaga v pravo smer reševanja. Rezultati testiranja so prikazani v tabeli 2. Prvi primer (vrstica) prikazuje vračanje iz napačne veje reševanja. Študent je pozabil opraviti stik s tretjo tabelo in se znašel v napačni veji. Sistem ga je pravilno opozoril z namigom, ki zahteva uporabo dodatne tabele. Druga vrstica tabele prikazuje preprosto operacijo spremembe pogoja v sklopu poizvedbe WHERE. Sprememba obenem spremeni poizvedbo študenta v pravilno rešitev. Tretja vrstica demonstrira dopolnitev vgnezdene poizvedbe.

Tabela 2: **Študija primerov za scenarij 2**

Idealna rešitev	Rešitev študenta	Namig
SELECT COUNT (*) FROM zaposleni z, oddelek o, lokacija l WHERE z.ID_oddelek = o.ID_oddelek AND o.ID_lokacija = l.ID_lokacija AND Regija = 'DALLAS' GROUP BY Regija;	SELECT COUNT(Z.ID_Zaposleni) FROM Zaposleni Z, Lokacija L WHERE Regija= 'DALLAS'	SELECT COUNT(z1.ID_zaposleni) FROM zaposleni z1, lokacija l1, oddelek o1
SELECT z.Priimek , n.Priimek FROM zaposleni z, zaposleni n WHERE z.ID_nadrejeni = n.ID_zaposleni;	SELECT z1.priimek, z2.priimek FROM zaposleni z1, zaposleni z2 WHERE z2.id_zaposleni = z1.id_zaposleni	SELECT z1.priimek , z2. priimek FROM zaposleni z1, zaposleni z2 WHERE z2.id_zaposleni = z1.id_nadrejeni
SELECT * FROM zaposleni WHERE Placa = (SELECT MAX(Placa) FROM zaposleni WHERE Placa < (SELECT MAX(Placa) FROM Zaposleni));	SELECT * FROM zaposleni WHERE placa = (SELECT max(placa))	SELECT * FROM zaposleni WHERE Placa = (SELECT MAX(Placa) FROM zaposleni)

6 SKLEP

V okviru raziskave je bil razvit adaptivni sistem za lajšanje učenja jezika SQL. Sistem uporablja znanje, skrito v preteklih poskusih reševanja nalog iz jezika SQL za konstrukcijo zbirke znanja, ki se uporablja za generiranje namigov. Za uporabo znanja ter napovedovanja naslednjega ugodnega stanja se uporabljajo odločitveni procesi Markova, ki omogočajo napovedovanje v negotovih razmerah. Dodatno lahko generiranje namigov pospešimo z idealnimi rešitvami učiteljev, ki služijo kot začetna zbirka znanja.

Preprosta evalvacija sistema iz petega razdelka kaže, da je sistem zmožen generirati uporabne namige in s tem vsaj do določene mere pospešiti učenje jezika SQL. Poleg scenarijev, opisanih v petem razdelku obstajajo okoliščine, v katerih sistem ni zmožen ponuditi uporabnega namiga. Razlog je v pomanjkanju zgodovinskih podatkov oziroma variabilnosti le-teh. Tipičen primer je kompleksna rešitev študenta, z mnogimi vgnezdenimi poizvedbami. Obstaja velika verjetnost, da nihče ni reševal naloge na podoben način, zato sistem ni zmožen opraviti ujemanja z boljšim stanjem. Namigi v takšnih primerih vodijo študenta nazaj v eno izmed začetnih stanj in potemtakem niso uporabni. Možnih izboljšav je več. Ena pomembnejših je zamenjava načina ujemanja med stanji. Trenutno ujemanje poteka z uporabo striktnega algoritma za primerjanje drevesnih struktur. Boljša alternativa bi bila, da bi določili ključne objekte iz vsake poizvedbe in nato opravili ujemanje med takšnimi objekti.

Morebitna dodatna slabost sistema je v generiranju korakov rešitve oziroma v predpostavki, da študentje rešujejo poizvedbo po sklopih. Kot je razvidno iz rezultatov testiranj, so namigi močno usmerjeni v reševanje po pristopu od zgoraj navzdol zaradi narave korakov rešitev. Zaradi omejenih zgodovinskih podatkov smo bili primorani sprejeti omenjeno predpostavko reševanja po sklopih. Če bi sistem uporabljali v prihodnosti, bi lahko zamenjali strategijo generiranja korakov rešitev s sprotnim pošiljanjem poizvedb na strežnik. Tako bi dobili bolj realne podatke o postopku reševanja študentov.

7 LITERATURA

[1] Aleven, V., McLaren, B. M., Sewall, J. (2009). 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, 2*(2), 64–78. doi:10.1109/tlt.2009.22.

[2] Anderson, J. R., Corbett, A. T., Koedinger, K. R., Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The journal of the learning sciences, 4*(2), 167–207.

[3] Barnes, T. M., Stamper, J. C., Lehman, L., Croy, M. (2008). A pilot study on logic proof tutoring using hints generated from historical student data. V R. S. J. d. Baker, T. Barnes in J. E. Beck (ur.). *Proceedings from EDM'08: The 1st International Conference on Educational Data Mining* (str. 197–201). Montreal, Quebec, Canada.

[4] Fournier-Viger, P., Nkambou, R., Nguifo, E. M., Mayers, A., Faghihi, U. (2013). A Multiparadigm Intelligent Tutoring System for Robotic Arm Training. *Ieee Transactions on Learning Technologies, 6*(4), 364–377. doi:10.1109/tlt.2013.27.

[5] Jarvis, M. P., Nuzzo-Jones, G., Heffernan, N. T. (2004). Applying machine learning techniques to rule generation in intelligent tutoring systems. V J. C. Lester, R. M. Vicari in F. Paraguaçu (ur.). *Proceedings from ITS 2004: Intelligent Tutoring Systems: 7th International Conference, proceedings* (str. 541–553). Maceió, Alagoas, Brazil: Springer.

[6] Martin, B. I. (2002). *Intelligent Tutoring Systems: The practical implementation of constraint-based modelling* (Doctoral dissertation). University of Canterbury: New Zealand.

[7] Matek, T. (2015). *Adaptive system for learning SQL language* (BSc). University of Ljubljana, Slovenia.

[8] Mitrovic, A. (1998). Learning SQL with a computerized tutor. *ACM SIGCSE Bulletin, 30*(1), 307–311.

[9] Mitrovic, A. (2010). Modeling Domains and Students with Constraint-Based Modeling. V R. Nkambou, J. Bourdeau, in R. Mizoguchi (ur.), *Advances in Intelligent Tutoring Systems* (str. 63–80). Berlin, Heidelberg: Springer Berlin Heidelberg.

[10] Mitrovic, A., Martin, B., Suraweera, P. (2007). Intelligent tutor for all: The constraint-based approach. *Ieee Intelligent Systems, 22*(4), 38–45. doi:10.1109/mis.2007.74.

[11] Mitrovic, A., Ohlsson, S. (2006). Constraint-based knowledge representation for individualized instruction. *Computer Science and Information Systems, 3*, 1–22.

[12] Mitrovic, A., Ohlsson, S., Barrow, D. K. (2013). The effect of positive feedback in a constraint-based intelligent tutoring system. *Computers & Education, 60*(1), 264–272.

[13] Parr, T. J., Quong, R. W. (1995). ANTLR – A Predicated-LL(k) parser generator. *Software-Practice & Experience, 25*(7), 789–810.

[14] Polson, M. C., Richardson, J. J. (2013). *Foundations of intelligent tutoring systems*: Psychology Press, L. Erlbaum Associates Inc.

[15] Prior, J. C., Lister, R. (2004). The backwash effect on SQL skills grading. *ACM SIGCSE Bulletin, 36*(3), 32–36.

[16] Stamper, J. C., Barnes, T. M., Croy, M. (2011). Enhancing the automatic generation of hints with expert seeding. *International Journal of Artificial Intelligence in Education, 21*(1–2), 153–167.

[17] Zhang, K., Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing, 18*(6), 1245–1262.

▪

Tadej Matek je leta 2015 diplomiral na področju računalništva in informatike na Fakulteti za računalništvo in informatiko Univerze v Ljubljani, kjer opravlja študij druge stopnje. Področja, ki ga zanimajo in s katerimi se ukvarja, so strojno učenje, analiza omrežij, inteligentni sistemi in agenti, poizvedovanje po podatkih ter razvoj informacijskih sistemov.

▪

Dejan Lavbič je leta 2010 doktoriral na področju računalništva in informatike na Fakulteti za računalništvo in informatiko Univerze v Ljubljani, kjer je zaposlen kot docent. Na raziskovalnem področju se ukvarja z inteligentnimi agenti, večagentnimi sistemi, ontologijami, poslovnimi pravili, semantičnim spletom, odkrivanjem plagiatorstva s pomočjo socialnih omrežij, kakovostjo informacij in naprednimi pristopi za porazdelitev podatkov. Sodeloval je pri številnih gospodarskih in raziskovalnih projektih s področja strateškega planiranja, metodologij razvoja informacijskih sistemov, uporabe inteligentnih agentov, avtomatizacije poslovnih procesov in obvladovanja ter porazdelitve velike količine podatkov.