T A R T U Ü L I K O O L
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut

Joel Edenberg

Automaatne andmetepõhine andmebaasi skeemade genereerimine

Magistritöö (30 EAP)

Juhendaja: Konstantin Tretjakov, M.Sc.

Autor: ..... "....." mai 2012

Juhendaja: ..... "....." mai 2012

Lubada kaitsmisele

Professor: ..... "....." mai 2012

TARTU 2012

Sisukord

Sissejuhatus

Nagu rahvatarkus ütleb, on inimene täpselt nii laisk kui võimalik ja nii töökas kui vajalik. Rutiinseid tegevusi üritatakse automatiseerida ning raskeid töid teha võimalikult palju masinate kaasabil. Infotehnoloogia on avanud uued võimalused paljude toimingute automatiseerimisel ja seetõttu on hakatud infosüsteeme kiiresti kasutama enamikes eluvaldkondades. Kui aga suurenevad võimalused, siis suurenevad ka inimeste ootused. Järjest enam eeldatakse, et süsteemid, millega suheldakse, on intuitiivsed ning sarnanevad mõnes mõttes inimesega – nad mäletavad kasutajat ning kohandavad talle personaalse töökeskkonna. Seega on andmete salvestus kesksel kohal igas tänapäevases tarkvarasüsteemis.

Iga uue tarkvaraprojekti alguses tuleb välja mõelda loodava rakenduse arhitektuur, sealhulgas ka andmemudel, mis võimaldaks salvestada soovitud detailsusel informatsiooni. Kuna aga üldjoontes soovitakse salvestada sarnast informatsiooni (süsteemi konfiguratsioon, süsteemi kasutajate andmed, objektid, millega süsteem manipuleerib jne), siis on ka andmemudelid rakendustel sarnased – vähemalt sama tüüpi rakendustel. Isegi kui uus loodav süsteem erineb kõikidest seniloodud lahendustest märkimisväärselt, on väga tõenäoline, et mingisuguses alamosas soovitakse salvestada sarnaseid andmeid. Seega tuleb iga tarkvaraprojekti alguses tihti läbida täpselt sama tööprotsess ja luua n-ö andmebaasi alusraamistik. Kuna tegemist on tüütu ning korduva tööprotsessiga, oleks mõistlik proovida automatiseerida andmebaasi mudelite genereerimist. Automaatselt genereeritud mudelit oleks seejärel võimalik kohandada konkreetse ülesande tarbeks. Kuna sellisel juhul ei peaks iga kord alustama täiesti algusest, võimaldaks automaatne andmebaasi mudelite genereerimine hoida kokku palju tarkvara arendajate tööaega.

Antud magistritöö eesmärgiks ongi uurida võimalusi andmemudelite ehk skeemade automaatseks genereerimiseks. Selleks pakume välja ühe konkreetse algoritmi ning arutleme lahenduse optimaalsuse üle. Väljapakutud lahendus töötab tingimuslikult sõltumatul tõenäosusmudelil ning komplekteerib lõppvastuse kokku vastates paljudele alamküsimustele eraldi. Nendeks küsimusteks on näiteks: kuidas on andmetabelid omavahel seotud, milliseid lisatabeleid oleks tarvis ning millised veerud peaksid tabelites leiduma? Kasutades tabelinimesid antakse algoritmile teada, millist informatsiooni soovitakse vastust esindavasse andmebaasi salvestada. Sisuliselt esindavad nimetatud tabelid andmeobjekte, mis peaksid lahendusskeemas kindlasti leiduma.

Kuna me ei suutnud leida ühtegi varem teostatud uurimustööd andmemudelite automaatse genereerimise teemadel, siis on välja pakutud lahendus suuresti eksperimentaalne. Nagu ka tööst hiljem selgub, esineb antud ülesande lahendamisel põhimõttelisi probleeme, mis oma olemuselt raskendavad automaatset andmemudelite genereerimist.

Peatükis 1 anname ülevaate töös kasutatavatest üldistest kontseptsioonidest. Peatükis 2 sõnastame täpselt uuritava probleemi ning pakume välja üldise lahenduse. Lisaks uurime lähemalt probleeme, mis tekivad skeemade automaatsel genereerimisel ning vaatleme alternatiivseid lahendusi. Väljapakutud algoritmi detailne realisatsiooni kirjeldus on esitatud peatükis 3. Peatükis 4 testime väljapakutud algoritmi tööd ning anname hinnangu tulemile.

Töö lisades on toodud:

- Terminoloogia
- Sisuhaldussüsteemide skeemad
- Skeema genereerimiseks kogutud andmed
- Algoritmi väljund
- Materjalide CD-plaat

Peatükk 1

Taustainformatsioon

1.1 Andmebaasid

Andmebaas on korrastatud andmete kogum. Andmed on tavaliselt organiseeritud peegeldamaks reaalses elus eksisteerivate objektide vahelisi seoseid. Andmebaasi võib lihtsustatud kujul vaadelda kui tabelit, mis koosneb ridadest ja veergudest. Andmebaasi täpsema ehituse kirjeldamiseks kasutatakse andmebaasi mudeleid ehk skeemasid.

Skeema on formaalses keeles kirjeldatud andmeebaasi täielik struktuur. See on n-ö detailne plaan andmebaasis leiduva informatsiooni ehituse kohta. Skeemad võib üldiselt jagada kolme kategooriasse: relatsioonandmebaasi skeemad, ontoloogilised andmebaasi skeemad (näiteks OWL, RDF) ning XML skeemad (näiteks XDR, XSD). Relatsioonandmebaasi skeema koosneb näiteks elementidest nagu tabelid ja tulbad. XML skeema koosneb aga elementidest ja atribuutidest.

SQL (*Structured Query Language*) on programmeerimiskeel, mis on loodud andmete haldamiseks relatsioonilistes andmebaasides. SQL programmeerimiskeeles defineeritakse andmebaasi skeemad kasutades lauseid (inglise keeles *statement*). Näiteks lause:

```
CREATE TABLE klient (Nimi char(50), Aadress char(50));
```

loob andmebaasi tabeli nimega “klient” millel on kaks veergu “Nimi” ja “Aadress”.

Andmebaaside haldamiseks on kasutusel mitmeid andmebaaside haldussüsteem. Levinumad on MySQL, PostgreSQL, Microsoft Access, Oracle, ja SQLite.

SQLite iseloomustab kompaktsus ja konfigureerimise lihtsus. Andmebaas salvestatakse ühte lokaalsesse faili ning selle kasutamiseks ei ole tarvis installeerida eraldi teenuse serverit.

1.2 Andmete töötlus

Regulaaravaldised (inglise keeles *regular expressions*) on sõne (tähtede ja sümbolite kombinatsioon), mis kirjeldab või langeb kokku mingi sõnede hulgaga vastavalt kindlatele süntaksireeglitele. Nad pakuvad kompaktset ning paindlikku võimalust sõnede valideerimiseks. Regulaaravaldisi saab kasutada näiteks sõnede otsimiseks pikemast tekstis või sisendi vastavuse kontrolliks etteantud mustrile. Näiteks regulaaravaldise ".as" jaoks valideeruvad muuhulgas sõned "kas", "las", "aas", kuid ei valideeru "jaa".

Regulaaravaldistest saab täpsema ülevaate raamatust [GL09].

Masinõpe on tehisintellekti haru, mille eesmärgiks on uurida algoritme, mis võimaldavad arvutitel automaatselt leida andmetest seoseid ja mustreid ning formuleerida need hilisemalt kasutatavaks teadmuseks. Masinõppe algoritmid võimaldavad teha empiiriliste andmete põhjal ennustusi ning otsuseid.

Masinõppe kasutusvaldkonnad võib üldistatult jagada kolmeks:

- Andmekaeve – varemkogutud info põhjal otsuste tegemine. Näiteks krediitkaardi tehinguid jälgides pettuste avastamine.
- Infosüsteemid, mida ei ole võimalik käsitsi programmeerida. Näiteks autonoomne auto juhtimine.
- Ise kohanduvad süsteemid. Näiteks uudisteluger, mis kuvab ainult kasutajat huvitavaid uudiseid.

Masinõppest on võimalik rohkem lugeda näiteks raamatust [Mit97].

Tõenäosuslik modelleerimine on tänapäevase tehisintellekti üks alustalasid. See võimaldab analüüsida suurtes kogustes andmeid väheste töökuludega. Tõenäosuslikku modelleerimist kasutatakse keeruliste reaalses elus eksisteerivate süsteemide üldistamiseks ning lihtsustamiseks. Selliste mudelite abil esindavaid süsteeme iseloomustab sõltuvus juhuslikest (või meile tundmatutest) parameetritest. Seega on ka vaadeldava süsteemi seisund tinglikult juhuslik ning selle kirjeldamiseks tuleks kasutada tõenäosusteooriat ning stohhastilisi protsessikirjeldusi.

Tõenäosuslikku modelleerimist on kasutatud näiteks järgmiste probleemide lahendamisel:

- Haiguste diagnoosimine sümptomite põhjal.
- Kasutaja eelistuste ennustamine varem sooritatud tegevuste põhjal.
- Rämpsposti filtrid.
- Evolutsioonipuude koostamine.

- Objektide asukoha hindamine radaril.

Tõenäosuslikust modelleerimiset on võimalik rohkem lugeda [Mit98], [Ble12].

Statistilised näitajad skeemade sarnasushinnangute analüüsimiseks.
Jaccardi koefitsienti kasutatakse andmehulkade sarnasuse ja mitmekesisuse hindamiseks. See arvutatakse kujul:

$$J = \frac{Sarnaste\ elementide\ arv}{Kõikide\ unikaalsete\ elementide\ arv}$$

Nimetatud koefitsient pakuti esmakordselt välja Paul Jaccardi poolt tema töös [Jac12].

Järgnevalt oletame, et me soovime ennustada sündmuse Y toimumist. Erinevad olukorrad, mis tekkivad prognooside ning tegelike tulemuste vahel saame esitada tabeliga:

	Tegelik Y = 1	Tegelik Y = 0
Prognoositud Y = 1	tp	fp
Prognoositud Y = 0	fn	tn

Seega saame tabelis leiduvad muutujad nimetada järgnevalt:
tp - Tõeliselt positiivne (inglise keeles *true positive*).
fp - Valepositiivne (inglise keeles *false positive*).
tn - Tõeselt negatiivne (inglise keeles *true negative*).
fn - Valenegatiivne (inglise keeles *false negative*).

Kasutades antud termineid saame defineerida statistilisi näitajad.
Saagis (inglise keeles *recall*) arvutatakse kujul:

$$Saagis = \frac{tp}{tp + fn}.$$

Täpsus (inglise keeles *presicion*) arvutatakse kujul:

$$Täpsus = \frac{tp}{tp + fp}.$$

Täpsema ülevaate saab näiteks raamatust [OD08].

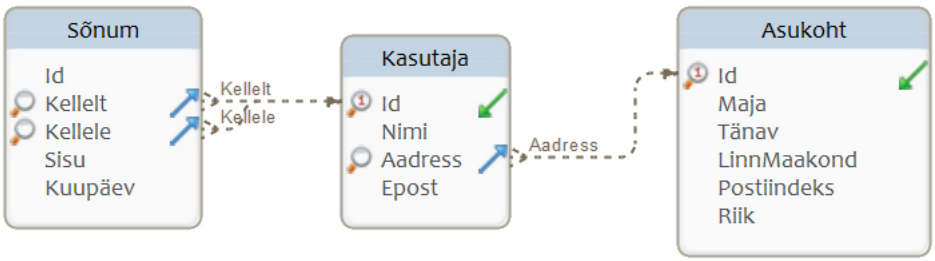
Peatükk 2

Skeema genereerimise algoritm

2.1 Skeema genereerimise vajadus

Andmebaas on tänapäevastes infosüsteemides kesksel kohal. Salvestusmeediumite kiire areng on muutnud riistvara nii odavaks, et tihti on mõistlik salvestada isegi rohkem informatsiooni kui esialgu plaanitakse kasutada. Andmete kogumine on seega väga odav ning samas avab võimalused tarkvara hilisemaks täiendamiseks ning keerulisema loogika loomiseks. Seega on andmebaaside loomine täiesti loomulik samm ning sellega puutub kokku peaaegu iga infosüsteemi loomisel.

Kuidas täpselt andmebaasi mudeleid ehk skeemasid esitleda ei ole üheselt määratletud. Skeemade n-ö defineerimiseks või kirjeldamiseks on küll kasutuses palju erinevaid formaalseid keeli (OWL, RDF, XML, SQL jne), kuid reeglina on võimalik andmemudel väljendada lihtsalt suunatud tsüklivaba graafina. Sellises graafis esindaksid tipud skeema elemente (nt relatsioonilistes skeemades tabelid ja veerud, XML'is aga atribuudid) ning suunatud servad erinevaid seoseid nende elementide vahel (nt sisaldumine/kuuluvus, võõrvõtmetena viitavad seosed jne). Selline graaf on tsüklivaba, kuna elemendid on omavahel hierarhilistes seostes. Antud töö raames vaatleme relatsioonilisi andmebaasi skeemasid (vt joonis 2.1), mis on defineeritud kasutades SQL programmeerimiskeelt.



Joonis 2.1: Relatsioonilise andmebaasi skeema esitatud graafilise mudelina.

Töö eesmärgina püstitatud probleemi aktualiseerimiseks vaatleme järgnevalt sar-

nase eesmärgiga infosüsteemide skeemasid. Valisime välja kolm veebliehekülgede sisuhaldust pakkuvat infosüsteemi - Joomla (`http://www.joomla.org/`), WordPress (`http://wordpress.org/`) ja Mambo (`http://mambo-foundation.org/`). Vaatleme lähemalt nimetatud infosüsteemide skeemade neid alamosasid, mis puudutavad kasutajaga seotud informatsiooni salvestamist (pildid skeemadest on toodud lisas 2 joonistel 1, 2 ja 3).

Nagu näeme on üldjoontes salvestatav informatsioon sarnane. Kõikides skeemades leidub tabel, mis salvestab kasutaja kui andmeobjekti, kusjuures selle kohta salvestatakse sarnaseid atribuute – nimi, kasutajatunnus, salasõna, e-posti aadress jne. Lisaks on võrdlemisi sarnased ka seosed teiste andmeobjektidega. Näiteks on kasutajaga mingil viisil seotud andmeobjekt, mis viitab postitustele või sõnumitele. Seega näeme, et sarnaste kasutusfunktsionaalsusega infosüsteemide andmebaaside skeemad on ka võrdlemisi sarnased.

Antud töö eesmärgiks ongi uurida kas ja kuidas oleks võimalik automaatselt andmebaasi mudeleid genereerida. Kuigi infosüsteemis kasutatavad skeemad ei ole kunagi täiesti identsed, leidub siiski väga suure tõenäosusega vähemalt mingisugune alamosa skeemast, mis on väga sarnane teiste sama valdkonna infosüsteemidega. Selline skeemade osaline sarnasus on tingitud asjaolust, et andmebaasis soovitakse üldjoontes salvestada siiski sarnast informatsiooni (nt kasutajad, sõnumid, andmeobjektid millega rakendus opereerib jne). Seega oleks mõistlik selline korduv tegevus automatiseerida ning lähtudes üldistest funktsionaalsetest vajadustest (nt "kasutajate haldamine") üritada andmebaasi mudeleid automaatselt genereerida.

2.2 Võimalikud lahendused

Enne konkreetsetest lahendustest rääkima asumist täpsustame üle kuidas kasutaja peaks algoritmile kirjeldama skeemat, mida ta soovib automaatselt genereerida. Nimelt otsustasime selleks kasutada väga minimalistlikus koguses informatsiooni, andes lahendusalgoritmidele ette ainult tabelinimed, mis peaksid lõpptulemis (vastusena tagastatavas skeemas) kindlasti olemas olema. Seega kasutaja peab olema suuteline identifitseerima ning nimetama olulisemad andmeobjektid, mille kohta soovitakse andmebaasis informatsiooni salvestada. Üldistatult võib öelda, et me üritame, tuginedes kasutajalt saadud informatsioonile, ennustada milline skeema oleks antud kontekstis kõige sobilikum ning seejärel see kasutajale tagastada.

Loomulikult toob selline lähenemine kaasa ka täiesti fundamentaalsel tasemel probleeme – nimelt ainuke kes tegelikult ka teab, milline skeema on reaalselt vastuseks sobilik, on kasutaja. Antud probleemi leevendamiseks võiksime lahendusalgoritmile ette anda rohkem taustinformatsiooni soovitud skeema kohta, kuid teatud piirides jääb probleem samaks. Meie algoritm ei ole täielikult teadlik teda ümbritsevast kontekstist ning puudub arusaam mille tarvis skeemat üritatakse luua. Seoses sellega puudub ka võimalus luua automaatselt arvutatav hinnangufunktsioon tulemusele. Meil ei ole võimalik kasutades hinnangufunktsiooni and-

maks vastustele jooksvalt hinnanguid ning jõudmaks seeläbi paremate tulemusteni. Sisuliselt puudub objektiivne kriteerium, mille abi oleks võimalik algoritmi poolt väljapakutud vastuseid jooksvalt hinnata ja algoritmi tööd suunata.

Skeemade automaatset genereerimist võib üldiselt käsitleda nii otsustusprobleemina (millised tabelid ning veerud peaksid lõppvastuses olemas olema) kui ka otsinguprobleemina (millised varemnähtud skeemasid esindaksid kõige paremat vastust). Otsustusprobleemi saab omakorda jagada reeglipõhisteks lahendusteks ning andmepõhisteks lahendusteks.

Reeglipõhiste lahenduse puhul tuleks defineerida hulk reegleid, mis kirjeldaksid kuidas skeema genereerimisel käituda. Näiteks oleks iga tabeli puhul mõistlik lisada veerg "ID", mis muudaks tabelisse salvestatavad andmeread unikaalseteks ning võimaldaks neile hiljem viidata. Kuid lisaks on vaja veel suurel hulgal mittetriviaalseid reegleid, väljendamaks millised lisatabeleid (nt kasutajate kohta infot salvestades oleks ehk mõistlik luua ka kasutajate õigusi puudutav lisatabel) ning tabelites leiduvaid veerge oleks tarvis. Tingimustes kus valdkond, milles skeemasid tahetakse genereerida on piisavalt väike ja alati sarnane, on reeglipõhised lahendused piisavad. Juhul kui aga skeemad esindavad väga paljusid erineva funktsionaalsusega infosüsteeme, tuleks selle asemel kasutada andmepõhiseid lahendus, nagu näiteks masinõpet. See võimaldaks leida andmetest tihtiesinevaid seaduspärasusi ning kasutada neid näiteks reeglite formuleerimiseks, mille abil vastus konstrueeritakse.

Otsingupõhine põhinev lahendus kasutab vastuse leidmiseks näidiseid skeemadest, mis pärinevad reaalselt kasutuses olevatest infosüsteemidest. Selline varemnähtud skeemade kogum moodustab andmepõhiste algoritmide teadmusbaasi - andmed mida kasutatakse vastuste leidmiseks. Konkreetsel juhul kasutame treeningandmetena skeemasid, mis on defineeritud kasutades SQL programmeerimiskeelt. Meiepoolt väljapakutud lahendusalgoritm kasutabki andmeotsingupõhist lähenemist, kuna see annab lootust luua piisavalt üldistava ning laialtkasutava süsteemi. Etteantud tabelinimede põhjal leitakse teadmusbaasist võimalikult sarnased skeemad ning nende põhjal pakkutakse välja sobilik lahend.

Piisavalt suure teadmusbaasi korral tekkib olukord, kus otsingupõhine lahendus suudab iga päringu põhjal tagastada vähemalt ühe sobiliku skeema. Seega sobiliku skeema genereerimiseks on kaks võimalust – leida otsingutulemuste seast mingisuguse objektiivsete kriteeriumite põhjal üks kõige sobilikum vaste või sobitada kõik otsingutulemused kokku üheks optimaalseks vastuseks.

Otsingutulemustest kõige sobilikuma vastuse leidmine aga sisuliselt kalduks vastuseid ülesobitama (inglise keelse *overfitting*) ning vastuste kvaliteet oleks tugevas sõltuvuses teadmusbaasi suurusest ning sinna sattunud andmetest. Isegi väga suure teadmusbaasi korral ei õnnestuks täiesti unikaalsete skeemade genereerimine ning puuduks üldistusvõime. Selline lahendus sooritaks hästi ainult varemnähtud skeemadele identsete vastuste genereerimisel ehk siis olukordades, kus infosüsteemid mille skeemasid genereeritakse on oma olemuselt väga sarnased.

Otsingutulemuste kombineerimine ühtseks vastuseks on aga juba masiõppe valdkonna probleem. Masinõppel põhinevad lahendused võimaldavad leida andmetest erinevaid üldistatud seoseid ning mustreid – õppida seaduspärasusi, mis võimaldavad oletada milliste tabelinime komplektide juurde sobivad millised vastust esindavad skeemad. Kuna antud lähenemine võimaldab paremaid tulemusi ka seninägemata skeemade genereerimisel, siis selles suunas kaldub ka meiepoolt väljapakutud lahendusalgoritm. Kasutame skeemade genereerimiseks vastuse tõenäosuslikku modelleerimist.

2.3 Probleemi ja selle lahenduse formuleering

Enne probleemi täpset sõnastus matemaatilise formuleeringuna, defineerime hilisemaks kasutamiseks järgmised mõisted:

Definitsioon 2.1. Tabel *on objekt, mille määratleb üheselt tema nimi ning selles sisalduvad atribuudid.*
Definitsioon 2.2. Tabeli atribuut *on objekt, mille määratleb tema nimi. Sisuliselt sümboliseerivad atribuudid tabelisse salvestatavaid parameetreid.*
Definitsioon 2.3. Atributeeritud tabeliks *nimetame tabelit, mis sisaldab vähemalt ühte atribuuti.*
Definitsioon 2.4. Võrdelisteks skeemadeks *nimetame skeemasid, mis sisaldavad täpselt samasid atributeeritud tabeleid.*
Definitsioon 2.5. Skeemade suuruseks *nimetame skeemas leiduvate tabelite arvu.*

Tähistame atributeeritud tabeleid kujul $t(a_1, a_2, ..., a_n)$, kus $a_1, a_2, ..., a_n$ tähistavad tabelis leiduvaid atribuute. Seega iga skeema y saame esitada kui jada selles leiduvatest atributeeritud tabelitest, st $y = (t_1(a_{1,1}, a_{1,2}, ...), t_2(a_{2,1}, a_{2,2}, ...), ...)$. Tabelinimede hulga X esitame kujul $X = (n_1, n_2, ...)$, kus element n_m tähistab m-inda tabeli nime. Nüüd saame sõnastada skeemade automaatse genereerimise probleemi formaalsel kujul järgnevalt:

Definitsioon (Skeema genereerimise ülesanne).
Leida funktsioon f : päring $\rightarrow y$, mis seab sisendina saadud päringule vastavusse sobiliku skeema y. Päringuna käsitleme etteantud tabelinimede hulka X.

Funktsiooni f leidmist vaatleme kui masinõppe probleemi, mille ülesandeks on sobiliku funktsiooni f automaatne õppimine etteantud treeningnäidete põhjal. Antud juhul on treeningnäideteks kogumik reaalselt kasutuses olevatest skeemadest. Funktsiooni f leidmiseks kasutame tõenäosuslikku lähenemist. Nimelt saame lahendusfunktsiooni poolt vastusena tagastatud skeema y sobilikkuse hinnangu esitada tõenäosuslikult. Kui y esindab ühte võimalikku vastusena tagastatud skeemat, siis tõenäosus, et just skeema y on parim vastus antud päringu korral, saab esitada kujul $P(y|päring)$. Seega peab funktsioon f vastusena tagastama skeema, millel selline tinglik tõenäosus on kõrgeim. Formaalselt saame lahendusfunktsiooni

esitada kujul $f(päring) = \text{argmax}P(y|päring)$.

Lahendusfunktsiooni leidmiseks teeme järgnevalt lihtsustava eelduse. Me oletame, et näidisandmetena kogutud skeemade andmestikku võib ettekujutada kui tõenäosuslikult sõltumatut ja ühtlase jaotusega (inglise keeles *independent and identically distributed*) andmestikku kõikidest skeemadest. Sellisel juhul on tinglikud tõenäosused leitavad kujul

$$P(y|päring) = \frac{\textit{Skeemade arv, mis on võrdelised skeemaga y ja vastavad päringule}}{\textit{Skeemade arv, mis vastavad päringule}}$$

On selge, et skeemades leiduvad tabelid ning ka tabelites leiduvad atribuudid on ilmselt omavahel seotud. Näiteks võivad mõned tabelid või tabeli atribuutide komplektid alati esineda üheskoos. Siinkohal teeme aga lihtsustava oletuse, eeldades edaspidises, et tabelid on omavahel tinglikult tõenäosuslikult sõltumatud. Antud oletus ei ole ilmselt korrektne, kuid tegemist on väga laialtlevinud lähenemisega, mis võimaldab arvutusi oluliselt lihtsustada.

Kuna skeema koosneb atributeeritud tabelitest, siis saame kirjutada, et

$$\begin{aligned}&P(skeema\ y|päring) = \\ &P(t_1(a_{1,1},...,a_{1,m}), t_2(a_{2,1},...,a_{2,m}),...,t_n(a_{n,1},...,a_{n,m}|päring),\end{aligned}$$

kus muutuja t_n viitab skeema y n-indale tabelitele ning muutuja $a_{n,m}$ viitavab n tabeli m-indale atribuudile. Kasutades nüüd tingliku tõenäosuse sõltumatuse eeldust saame skeema esinemise tõenäosuse väljendada kujul

$$\begin{aligned}&P(t_1(a_{1,1},...,a_{1,m}), t_2(a_{2,1},...,a_{2,m}),...,t_n(a_{n,1},...,a_{n,m})|päring) = \\ &P(t_1(a_{1,1},...,a_{1,m})|päring) \cdot ... \cdot P(t_n(a_{n,1},...,a_{n,m})|päring).\end{aligned} \tag{2.1}$$

Seega skeema esinemise tõenäosus on väljendatud iga temas leiduva atributeeritud tabeli esinemise tõenäosuste korrutisena. Järgmiseks eeldame, et ka tabelite atribuudid on tinglik tõenäosuslikult sõltumatud. Seega saame valemi (2.1) lahti kirjutada kujul

$$\begin{aligned}&P(t_1(a_{1,1},...,a_{1,m})|päring) \cdot ... \cdot P(t_n(a_{n,1},...,a_{n,m})|päring) = \\ &P(t_1|päring)P(a_{1,1}|t_1)P(a_{1,2}|t_1) \cdot ... \cdot P(a_{1,m}|t_1) \cdot ...\cdot \\ &\cdot P(t_n|päring)P(a_{n,1}|t_n)P(a_{n,2}|t_n) \cdot ... \cdot P(a_{n,m}|t_n),\end{aligned}$$

Kokkuvõtvalt saame skeema vastuseks sobimise tõenäosuse väljendada kujul

$$P(skeema\ y|päring) = \prod_{i\,=\,1}^{y\ tabelite\ arv} P(t_i|päring) \prod_{i\,=\,1}^{y\ tabelite\ arv} P(a_{i,1},...,a_{i,m}|t_i). \tag{2.2}$$

Siinkohal juhime tähelepanu asjaolule, et tõenäosuste korrutamise tõttu eelistab antud mudel väheste tabelite ning atribuutidega skeemasid. See aga pole ilmselt soovitud tulemus. Kuna aga vastusena tagastatavate tabelite koguarv on ettemääratud (etteantud tabelite ja kasutaja poolt soovitud lisatabelite koguarv),

siis valemi (2.2) esimese poole arvutamine probleeme ei valmista. Valemi teise poole (tõenäoliste atribuutide leidmise) saame aga esitada kujul

$$P(a_1, a_2, ..., a_m|tabel\ t) = \begin{cases} 1 & \text{kui iga } n \text{ korral } P(a_n|tabel\ t) \geq k, \\ 0 & \text{teistel juhtudel.} \end{cases}$$

Muutuja k on etteantud tabeli atribuutide lisamise lävend. Selle võib määrata nii kasutaja kui ka algoritmi siseselt.

Sellisel kujul arvutatud tõenäosust võib vastuse leidmiseks kasutada mitmeti. Meiepoolt kasutatud maksimaalse tõenäosusega skeema valimine on ainult üks võimalikest variantidest. Näiteks võiks vastuse valida juhuslikult skeemade hulgast, mis on piisavalt tõenäolised. Või kombineerida kõik piisavalt tõenäolised skeemad mingil viisil ühtseks vastuseks.

Edasi vaatleme täpsemalt kuidas väljapakutud algoritm vastuse komplekteerib. Oletame, et kasutaja soovib genereerida skeemat, mis sisaldab näiteks tabeleid nimedega t_1, t_2 ning t_3. Sellisel juhul leiame esiteks kõik skeemad, kus etteantud tabelid esinevad. Tagastatud skeemadest kogutakse kokku kõik võimalikud unikaalsed tabelinimed ning iga sellise tabeli t_p puhul hinnatakse kui suur on tõenäosus, et tabelite t_1, t_2 ja t_3 olemasolul leidub ka tabel t_p. Seega leitakse tinglik tõenäosus

$$P(t_p|t_1, t_2, t_3). \tag{2.3}$$

Vastusena tagastatakse kõik etteantud tabelid ja lisaks kasutaja poolt soovitud kogus kõige tõenäolisemaid lisatabeleid.

Peale vajalike tabelite leidmist jätkatakse tööd tabelite atribuutide (veergude) leidmisega. Iga tabeli puhul leitakse selles kõige sagedamini esinevad atribuudid. Tähistame tabelit, millele atribuute otsime tähega t. Iga atribuudi a_n puhul leiame tõenäosuse $P(a_n|t)$. Tagastame ainult kasutaja poolt määratud tõenäosuslävendi ületanud atribuudid.

2.4 Võõrvõtme heuristika

Lisaks tabelinimede informatsioonile on meil juurdepääs ka andmebaasides leiduvatele tabelitevahelistele seostele (võõrvõtmetega viitamine). Antud informatsiooni saame kasutada tabelite tinglike tõenäosuste modelleerimisel. Kirjeldatud heuristika kasutamine on valikuline.

Oletame, et meil on etteantud tabelid t_1, t_2 ja t_3. Tabeli t esinemistõenäosuse saame seega väljendada valemi 2.3 abil. Lisaks leiame tõenäosuse $P_{parandus} = P(t|t_1, t_2, t_3, V(t))$, kus $V(t)$ sümboliseerib asjaolu, et tabel t on võõrvõtme abil seotud vähemalt ühe tabeliga t_1, t_2 või t_3. Paranduse tulemusena tagastame tabeli t esinemistõenosuse kujul $P = P(t|t_1, t_2, t_3) + P_{parandus}$. Seega näiteks tabelid, mis alati on võõrvõtmete abil seotud mingil viisil etteantud tabelitega, saavad enda

esinemise tõenäosuse kaks korda suurema kui sama sagedasti esinevad mitteseotud tabelid.

Kindlasti annaks võõrvõtmetega seotuse informatsiooni kasutada ka teistsugustel viisidel, kuid eelkirjeldatud lähenemine tundus lihtsasti realiseeritav ning kohati parandas algoritmi töötulemust märgatavalt.

2.5 Tõenäosuslik skeemade genereerimine

Väljapakutud algoritmi realisatsioon kasutab näidisandmetena SQL programmeerimise keeles kirjeldatud skeemade definitsiooni faile. Iga skeema kohta kogutakse järgmised andmed: skeemas leiduvate tabelite arv, tabelite nimed, igas tabelis leiduvate veergude nimed, iga veerurgu defineeriv konkreetne SQL süntaks ning tabelite vaheliste võõrvõtmetega viitamiste olemasolu. Antud informatsioon salvestatakse teadmusbaasi, millesse algoritm hakkab hiljem vastuste genereerimiseks päringuid teostama.

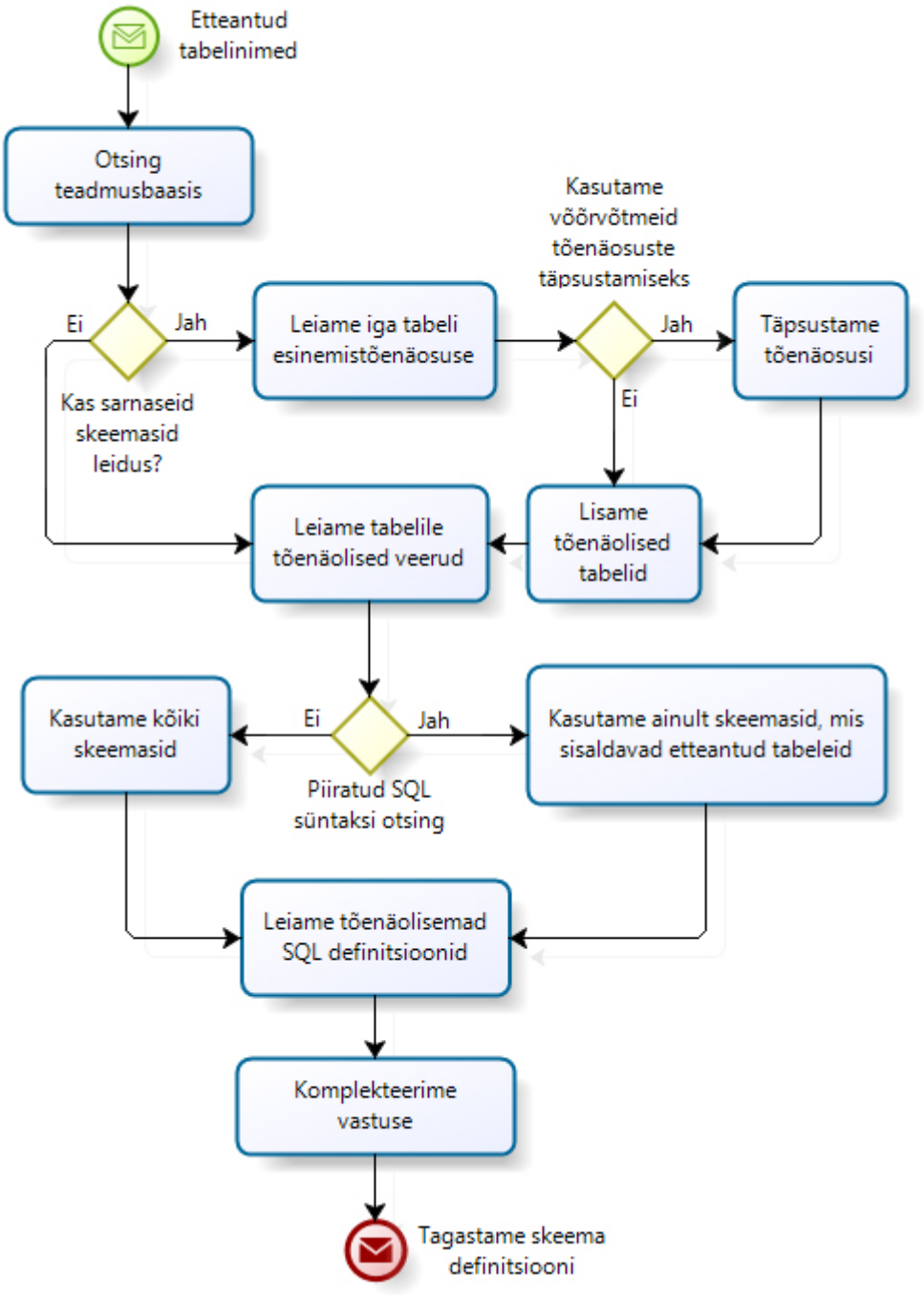
Algoritmi töö alustamiseks tuleb kasutajal ette anda üks või rohkem tabelinimesid, mis peavad lõppvastust esindavas skeemas kindlasti olemas olema. Põhimõtteliselt teavitatakse seeläbi algoritmi andmeobjektidest, mida kindlasti salvestada soovitakse ja seeläbi antakse vihje millist tüüpi skeemat soovitakse genereerida.

Edasi sooritab algoritm päringu teadmusbaasi leidmaks kõik sellised skeemad, mis sisaldavad etteantud tabeleid. Vastusena tagastatud skeemade hulgas leitakse iga tabeli kohta selle esinemise tõenäosus. Tõenäosuste leidmisel eeldatakse, et tabelite esinemised on tõenäosuslikult sõltuvad ainult etteantud tabelitest. Kasutaja soovi korral kasutatakse tõenäoliste lisatabelite hinnangute rafineerimiseks peatükis 2.4 kirjeldatud lisameetodit. Tõenäosuslik lävend, millest alates tabel ka lõppvastusesse lisatakse, võib olla määratud nii kasutaja poolt kui ka algoritmiliselt (nt etteantud tabelite arvust poole väiksema koguse lisatabeleid). Juhul, kui etteantud tabelinimede põhjal ei suudeta leida ühtegi skeemat (teadmusbaasis puudub näide skeemast, mis võimaldaks kõiki soovitud andmeobjekte salvestada) kasutatakse lõppvastuse genereerimisel ainult etteantud tabelid ning jätkatakse tööd.

Järgmiseks leitakse kõikidele vaadeldavatele tabelitele (nii etteantud kui leitud lisatabelitele) vajalikud veerud. Selleks leitakse teadmusbaasist kõik etteantud tabelite nimelised tabelid ning salvestatakse neis leiduvad veerunimed. Kõige sagedamini esinevad veerud lisatakse vaadeldavale tabelile. Jälle on lävendi, mille ületamisel veerg tabelisse lisatakse, määramiseks kaks võimalust - algoritmiliselt (nt juhuslikult) või kasutaja poolt. Veerge defineeriva SQL süntaksi leidmiseks on võimalik kasutada otsingut üle kõikide skeemade, kus leidub tabel milles sisaldub hetkel vaadeldav veerg, või otsingut üle nende skeemade, kus leiduvad kõik etteantud tabelid.

Viimase sammuna komplekteeritakse kogutud informatsioon tagasi kokku ning moodustatakse SQL programmeerimiskeeles kirjeldatud skeema täielik definitsioon.

Algoritmi tööst visuaalse ülevaate saamiseks on joonisel 2.2 kujutad algoritm ka tööprotsessi mudelina.



Joonis 2.2: Algoritm esitatud tööprotsessi mudelina.

Peatükk 3

Algoritmi realisatsioon

Antud peatükis esitleme detailselt ühte väljapakutud algoritmi konkreetset realisatsiooni. Tööga on kaasas ka lähtefailid, kust on võimalik näha konkreetse realisatsiooni lähtekoodi. Algoritmi realiseerimiseks kasutasime Pythoni programmeerimiskeele versioon 3.2. Kuna välja pakutud lahendus töötab SQL skeemadefinitsiooni failidega, siis oli tarvis head tuge sõnetöötluse teostamiseks. Lisaks on Pythonit produktiivne ning meeldiv kirjutada, kuna see on väga kõrgetaseme keel. Vastavalt objektidele, millega rakendus manipuleerib, jaguneb antud peatükk esimene pool kolmeks alamosaks: andmete töötlemine, andmete salvestamine ja päringu töötlemine (andmete pärimine ning tõenäosuslike vastususte konstrueerimine). Peatüki lõpus analüüsime tehtud valikuid, tutvustame põgusalt loodud kasutajaliidest ning kirjeldame algoritmi tööks tehtud eeldusi.

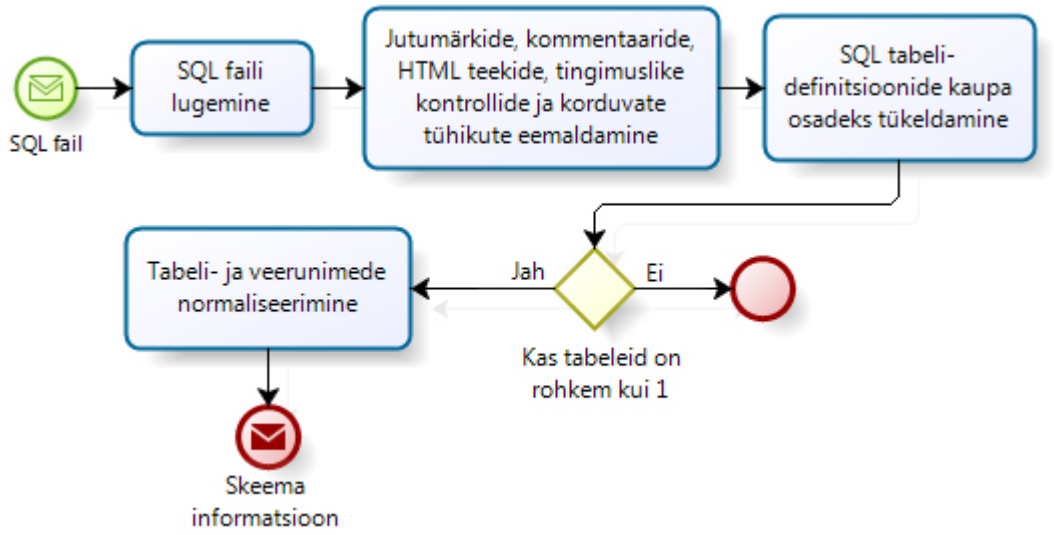
3.1 Andmete kogumine ja töötlemine

Tõenäosusliku lähenemise kasutamisel vastuste leidmiseks on tarvis suurt kogust näidisandmeid ehk teadmusbaasi. Esimese sammuna tuli lahendada andmete kogumise probleem. Otsustasime kasutada relatsioonilise andmebaasi defineerimise mudeleit - SQL andmebaasi loomise failid. Kuna enamus otsingumootoreid ei luba enda kasutamistingimustes mitte mingisugusel kujul automaatseid päringuid, kujunes andmete kogumine võrdlemisi vaevaliseks. Parimaks lahenduseks osutus Google poolt pakutud AJAX'i (*Asynchronous JavaScript and XML*) rakenduste liidese kasutamine, mis võimaldab korraga tagastada kuni kaheksa vastet. Suurendamaks võimalusi leida iga otsinguga uusi ning unikaalseid skeemasid, lisasime otsingut piiravate võtmesõnade hulka kahest tähest koosneva sõne (juhuslikud kahe sümboli pikkused permutatsioonid tähestikust).

Saadud otsingutulemuste vastuste URLid (*Uniform resource locator*) avatakse ning salvestatakse lokaalselt. Salvestatud failide nimeks valitakse 35 sümbolit URLi lõpust ning saadud sõnest eemaldatakse seejärel viis viimast sümbolit (väl-

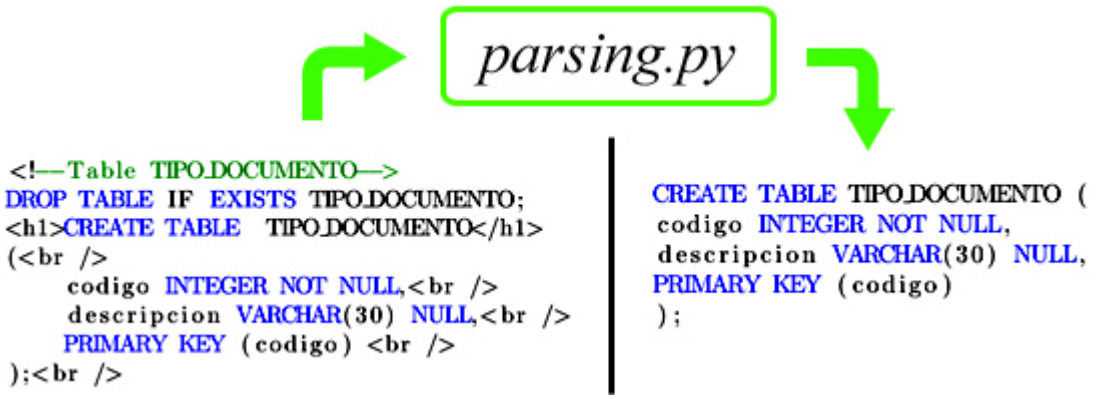
timaks dokumenditüübi laiendi kasutamist). Kõikide vastete salvestamisel kasutatakse ülekirjutamise režiimi ja seega sama nimega failide salvestamisel kirjutakse olemasolev üle. Sellisel viisil skeemafailide kogumine garanteerib vähemalt mingis ulatuses nende unikaalsuse. Otsingutulemuse vastete URL aadressid peaksid erinema väga suures ulatuses, arvatavasti isegi domeeninime võrra, et kaks skeemat saaksid mõlemad eraldi salvestatud.

Kogutud andmebaasi skeemade süntaksi analüüsimiseks (inglise keeles *parsing*) kasutame regulaaravaldistel põhinevat lähenemist. Kuna Python 3.2 jaoks ei ole veel ühtegi head SQL süntaksianalüsaatorit loodud ning antud ülesande lahendamiseks ei olegi vaja täieliku SQL struktuuripuu mõistmist, siis saigi otsustatud süntaksi töötlemine iseseisvalt realiseerida. Siinkohal parafraseerin Jamie Zawinski ütlust, et mõned inimesed kohates probleeme mõtlevad: "Mul tuli hea idee! Ma lahendand selle probleemi kasutades regulaaravaldisi."ja nüüd on neil juba kaks probleemi. See esialgu naljakana näiv ütlus osutus kahjuks vägagi tõeseks, kuna paljud skeemade definitsioonid olid kirjeldatud kasutades ebastandardset süntaksit. Seega tuli regulaaravaldisi kirjutades arvestada väga paljude erijuhtudega ning pidevalt tulemustes vigu leides neid täiendada. Skeemafailide süntaksianalüüsimiseks kasutatav kood asub failis nimega *parsing.py*. Joonisel 3.1 on kujutatud andmetöötlusel tehtavad sammud tööprotsessi mudelina.



Joonis 3.1: Andmetöötlus tööprotsessi mudelina.

Peale SQL skeemafaili sisselugemist eemaldatakse regulaaravaldiste abil sisendist ebaoluline müra nagu näiteks jutumärgid, SQL kommentaarid, HTML jne (joonis 3.2). Seejärel tükeldatakse sisend tabelite kaupa osadeks ning iga tabel töödeldakse eraldi.



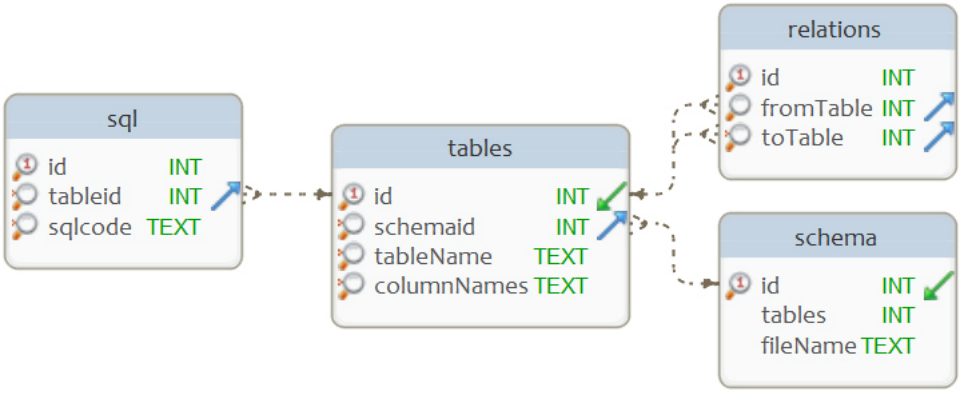
Joonis 3.2: Esmane SQL süntaksi töötlemine

Kui skeemas leidub vähem kui kaks tabelit, siis ignoreeritakse antud skeemat. Selline lähenemine tagab, et igasugused fiktiivsed skeemad ei sattuks algoritmi lõpliku teadmusbaasi. Ainult ühte tabelit sisaldavad andmemudelid on pigem erijuhud ning ei oma meiejaoks erilist väärtust. Iga skeemas leiduva tabeli puhul tükeldatakse antud tabelidefinitsioon selliselt, et veergude definitsioonid jäävad eraldi sõnedena. Veergude definitsioonidest otsitakse võõrvõtmete abil viitamisi teistele sama skeema tabelitele ning leitud seosed salvestatakse. Lisaks salvestatakse kõikide veergude nimed ning vastavad konkreetsed SQL definitsioonid. Lähtekoodis on antud funktsionaalsus realiseeritud failis *parsing.py* meetodites *processing()* ning *tableContent()*.

Kuna antud lahendus töötab suuresti kasutades ainult tabelite ning veergude nimesid, siis osutus suureks probleemiks nimedest genereeritud variatsioonid skeemade lõikes. Paljudes projektides lisatakse näiteks tabelinimede algusesse projekti initsiaalid. Või lisatakse tabeli veergude nimedele mingid sümbolid viitamaks konkreetsele tabelile. Sellised lahendused aga tekitavad olukorras, kus väga paljud sõned võivad sisuliselt tähistada küll sarnast andmeobjekti, kuid sõnede võrdluse seisukohast on nad täiesti erinvad. Seoses sellega sai teostatud prefiksite eemaldus nii tabeli kui ka veergude nimedest. Kui vähemalt 60% ühe skeema tabelinimedel või ühe tabeli veerunimedel esineb rohkem kui kahest sümbolist koosnev ühine prefiks, siis see lihtsalt eemaldatakse. Antud lahendus ei paku küll lõpliku vastust eelmainitud probleemile, kuid enamikel juhtudel on sellisel kujul nimede normaliseerimine siiski piisav. Sarnase tulemuse annaks ka sõnatüvede leidmine (inglise keeles *stemming*). Kahjuks ei ole selleks aga Python 3.2 jaoks veel loodud ühtegi laialtlevinud sõnatüvede leidmise teeki. Lähtekoodis on antud funktsionaalsus realiseeritud failis *parsing.py* meetodites *findPrefix()* ning *removePrefix()*.

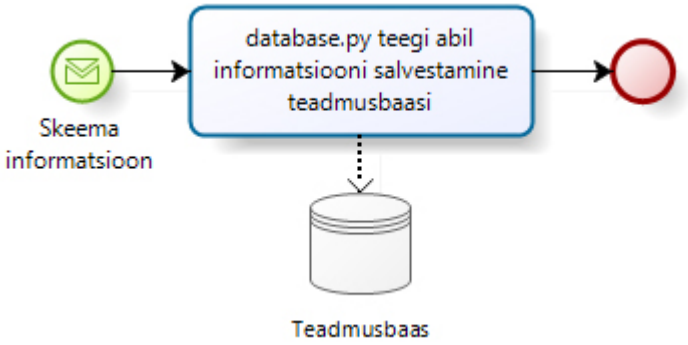
3.2 Andmete salvestamine

Iga näidisandmetena leitud skeema kohta salvestatakse teadmusbaasi hulgaliselt informatsiooni: tabelite arv, tabelite nimed, veergude nimed, tabelite vahelised seosed ning konkreetseid veerge defineeriv SQL süntaks. Joonisel 3.3 on kujutatud teadmusbaasi andmemudelit, mis võimaldab kõik eelnimetatud info salvestada.



Joonis 3.3: Teadmusbaasi andmebaasi mudel

Joonisel tähistavad veerunimede kõrval paremal asuvad sinised nooled võõrvõtmetega viitamist teistele tabelitele ning rohelised nooled märgivad veerge millele viidatakse. Suurendusklaasi kujutised sümboliseerivad asjaolu, et antud veerg on indekseeritud. Kuna tõenäosuslike küsimustele vastamiseks tuleb teha päringuid väga paljude kitsendavate parameetritega, siis annab indekseerimine suure jõudluse kasvu. Samas näidisandmeid lisatakse harva ning andmete salvestamise protsess toimub enne kui kasutaja hakkab teostama päringuid. Seega indekseerimisele kuluv lisaaeg ning mälumaht ei too kaasa erilisi negatiivseid kõrvalmõjusid.



Joonis 3.4: Andmete teadmusbaasi salvestamine tööprotsessi mudelina.

Konkreetse realisatsiooni puhul salvestatakse teadmusbaas SQLite andmebaasi, kuna see ei vaja lisateenuste paigaldamist ega seadistamist. Teadmusbaasi struktuuri defineerimine ning andmebaasi liidese loomine on realiseeritud failis *database.py*.

3.3 Päringu töötlemine

Joonisel 2.2 välja toodud tööprotsessi mudel illustreerib algoritmi töö käiku. Joonisel 3.5 on abstraktselt kujutatud päringute tegemisel tehtavaid samme.

Etteantud tabelid Töö alustamisel annab kasutaja ette nimekirja tabelinimedest, mis peavad lõppvastuses kindlasti olemas olema. Seejuures saab kasutada üldistavat süntaksit, mille abil laiendada võimalike sobilike tabelite nimesid. Märgend "%" abil tähistame suvalist arvu suvalisi sümboleid (regulaaravaldisena".*"). Seega näiteks tabelinimi "user%" viitab kõikidele nimedele, mis algavad sõnega "user" ning nimi "%user" viitab kõikidele nimedele, mis lõppevad sõnega "user". Kasutades sellist lähenemist saab kasutaja laiendada otsingut, eristamata seejuures nimesid võimalike prefiksite või sufiksite olemasolust.

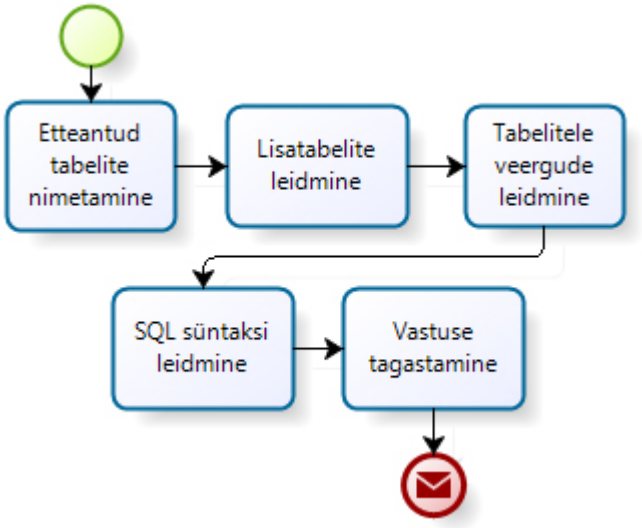
Lisatabelid Kasutaja määrab mitut võimalikku lisatabelit soovitakse vastusesse lisada. Seda arvu saab kasutaja korrigeerida peale väljapakutud tabelite ning nende esinemise tõenäosuste nägemist. Sellise lahenduse kasuks sai otsustatud seetõttu, et lisatabelite tõenäosused võivad erinevate skeemade genereerimiste raames kõikuda väga oluliselt. Kasutaja oleks sunnitud suhteliselt huupi arvates hindama, milline peaks olema lisatabelite tõenäosuslik lävend, et näiteks kaasta vähemalt üks lisatabel lõppvastusesse. Kasutaja seisukohast tundub loogilisem, et ette tuleb anda mitut lisatabelit soovitakse ja nähes võimalike lisatabelite tõenäosusi saab teha juba informeerituma otsuse nende vajalikkuse kohta.

Tõenäoliste lisatabelite leidmine toimub vastavalt peatükis 2.3 kirjeldatule. Juhul kui ei leidu ühtegi skeemat, mis sisaldaks etteantud tabeleid (soovitakse genereerida täiesti unikaalset andmemudelit teadmusbaasi mõttes), siis lisatabeleid ei lisata ning tööd jätkatakse ainult etteantud tabelitega. Lähtekoodis on antud funktsionaalsus realiseeritud failis *requests.py* meetodis *findProbableAdditions()*. Valikulise lisasammuna on võimalik eelmises etapis leitud tabelite esinemise tõenäosusi rafineerida kasutades peatükis 2.4 kirjeldatud heuristilist parandust. Kirjeldatud paranduse realisatsioon asub failis *requests.py* meetodis *adjustWithRelations()*.

Tabeli veerud Järgmiseks leitakse kõikide tabelite (nii kasutajapoolt etteantud kui ka leitud lisatabelite) jaoks nendes sagedasti esinevad veerud. Iga tabeli puhul saab kasutaja eraldi määrata protsentuaalse lävendi, mille ületamisel veerg tabelisse ka lisatakse. Tõenäosused on normaliseeritud selliselt, et kõige sagedamini esineva veeru sageduses loetakse 100% ning kõik ülejäänud tõenäosused arvutatakse lähtuvalt sellest. Selline lähenemine on õigustatud, kuna igas mõtet omavas tabelis peaks olema vähemalt üks veerg. Samuti aitab selline tõenäosuste normaliseerimine vältida olukordi, kus erinevate tabelite lõikes peaksid lävendid erinema suurusjärgu võrra lihtsalt selleks, et lisatavate veergude lõpparv oleks ligikaudselt võrdne.

SQL süntaks Seejärel otsitakse teadmusbaasist iga sagedasti esineva veeru kohta kõige tüüpilisem SQL definitsioon. Siinkohal saab kasutaja valida kahe erineva lahenduskäigu vahel. Esimese variandina kasutatakse süntaks leidmiseks kõiki teadmusbaasi tabeleid, millel on hetkel töös oleva tabeliga (tabel mille veergude süntaksit otsitakse) sama nimi. Teise variandina vaadatakse ainult neid tabeleid, mis asuvad skeemades, kus on olemas kõik kasutaja poolt etteantud tabelid. Ehk siis esimesel juhul leitakse täiesti üldine definitsioon, mis vastab vaadeldavale tabelinimele ning selles leiduvale veerule ning teisel juhul lisatakse kitsendus, et leitud skeemades peavad eksisteerima ka kõik teised kasutaja poolt etteantud tabelid. Esimesl juhul tagastatakse üldistatud veergude definitsioonid ning teisel juhul spetsiifilised definitsioonid, mis vastavad ainult etteantud alamgrupi skeemadele. Veerge defineeriva SQL süntaksi leidmine on realiseeritud failis *requests.py* meetodis *returnColumns()*.

Tagastatav vastus Viimase sammuna komplekteeritakse leitud informatsioon tagasi kokku lõppvastuseks. Vastusena tagastatakse skeemat defineerivad SQL lausesd. Lisas 3 on toodud näide informatsioonist, mida prototüüp skeema genereerimiseks kogub (eelviimane samm enne SQL süntaksi kokku komplekteerimist). Algoritmi prototüübi realisatsioon asub failis *implementation.py*. Programmi töö lõppedes salvestatakse tulemus samasse kataloogi, kus asus käivitatud *implementation.py*, faili nimega *out.sql*. Siinkohal juhime tähelepanu asjaolule, et antud realisatsioon on mõeldud ainult algoritmi testimiseks - üksikute skeemade genereerimiseks. Väljapakutud algoritmi mugavamaks kasutamiseks tuleks seda teha peatükis 3.5 kirjeldatud graafilise kasutajaliidese vahendusel.



Joonis 3.5: Päringute töötlemine tööprotsessi mudelina.

3.4 Realisatsiooni analüüs

Andmebaas Meie poolt väljapakutud lahendus vaatleb ainult relatsioonilisi andmebaase, jättes kõrval näiteks XML abil defineeritud andmemudelid. Esitledes skeemasid graafina, oleks võimalik sama algoritmi raames vaadelda ka skeemasid, mis on defineeritud kasutades teistsuguseid formaalseid keeli. Selline lahendus võimaldaks korraga vaadelda suuremat hulka erinevate skeemade näidiseid ja seeläbi tagastada ilmselt ka paremaid vastuseid.

SQL skeemade kiiremaks ja efektiivsemaks süntaksi analüüsimiseks võiks kasutada struktuuripuude ehitamist. Selline lahendus võimaldaks ilmselt paremini tõlgendada isegi ebastandardset süntaksit kasutavaid skeema-definitsioone ja seeläbi suurendada näidisandmete hulka.

Andmete kvaliteet Algoritmi üldise töökvaliteedi parandamiseks võiks rakendada reeglipõhiseid kontrollmeetodeid teadmusbaasi sisetavatele andmetele (nt kontrollida, et veerge defineeriv SQL on standarditele vastav). Selline lisakontroll võimaldaks piirata müra sattumise tõenäosust teadmusbaasi.

Kuna väljapakutud algoritm töötab kasutades ainult tabelinimesid, siis muudab see antud lahenduse väga tundlikuks nimede erinevatest variatsioonidest. Kuna sarnaseid objekte on võimalik nimetada väga paljudel erinevatel viisidel, siis on keeruline algoritmiliselt need objektid identifitseerida ning seejärel grupeerida. Näiteks kasutajate kohta infot salvestavad tabelid võivad kanda nime “user”, “users”, “usr”, või hoopis “member”. Lisaks võivad nimed olla erinevates keeltes ning tihti kasutatakse ka tabeli- ja veerunimede ees erinevaid prefikseid.

Antud probleemi leevendamiseks sai muuhulgas proovitud ka n-ö sõnaraamatu kontseptsiooni. Lühendid ning sarnase tähendusega sõned asendatakse mingisuguse üldlevinud versiooniga. Kuid selline lahendus toob ilmselt kaasa uusi probleeme. Prooviks kasutasime käsitsi loodud sõnaraamatut sünonüümidest. Ilmselgelt pole aga antud lahendus mõistlik ning skeemade arvukuse kasvamisel tuleks sõnaraamatu loomine automatsieerida. Kuid ka see ei oleks ilmselt triviaalne ülesanne, kuna sarnaseid andmeobjekte saab nimetada mitmeti ja samas on paljudel sõnadel mitu tähendust. Lisaks võib nimede teisendamisel tekitada nimedest dublikaate sama skeema või tabeli raames. Näiteks teisendades tulpade nimesid võiks küll olla mõistlik asendada sõned “gid”, “groupid”ja “group_id”sõnega “group”, kuid ainult juhul kui samas tabelis ei leidu samanimelist veergu. Kuna nimede unifitseerimise ülesanne osutuks väga mahukaks probleemiks, siis antud töö raames sellega rohkem ei tegeletud. Sünonüümide leidmise teemadel varem tehtud uurimustöö [WZ03].

Eelmise mõttekäigu edasiarendusena võiks proovida ka nimede täielikku tõlkimist teise keelde. Hetkel moodustavad teadmusbaasi inglise keelsete andmeobjekti nimedega skeemad. Kui me sooviksime aga skeemade genereerimisel nimed ette anda näiteks eesti keeles, siis tuleks sõnaraamatu kontspetsiooni veelgi laiendada ning

lisada soovitud keelte tugi. Tõlkimisel esineksid arvatavasti sarnased probleemid, mis tekkivad igasugu masintõlke realiseerimistel. Antud kontekstis peaks küll tegelema ainult masintõlke alamosaga, kuna muretsema ei pea keele grammatika pärast.

Jõudlus Väljapakutud algoritmi jõudluse parandamiseks võiks proovida SQL päringute optimeerimist. Hetkel kasutatakse otsingutel suurel hulgal kitsendavaid parameetreid. Olenevalt andmebaasi automaatsest päringuteoptimeerimise realisatsioonist, võivad teistsugused lahendused osutuda kiiremaks. Ühe võimaliku variandina saaks lõppvastuse kombineerida erinevate alapäringute liitmisel (kasutades näiteks erinevat tüüpi "JOIN"või "UNION" operaatoreid). Samuti võiks salvestada vahetulemusi hilisemaks taaskaustamiseks. Lisaks võiks kaaluda teiste andmebaasi mootorite kasutamist (nt MySQL või PostgreSQL). Oma ehituse poolest on SQLite küll väga mugav kasutada, kuid keerukamad lahendused ilmselt võimaldaksid suuremat jõudlust.

Üldised tähelepanekud Juhul kui teadmusbaasis ei leidu ühtegi etteantud tabelinimedega skeemat, siis lisatabeleid ei lisata. Tööd jätkatakse ainult etteantud tabelitega ning skeemat ei laiendata. Ilmselgelt on selline hetkelahendus piirav. Ühe võimaliku täiendusena saaks skeemasse lisada tabeleid kasutades mingit *Naive Bayes*i laadset tehnikat. Ehk siis vaadelda tabeleid tõenäosuslikult täiesti sõltumatult. Seega lisatabelite leidmisel saaks otsida kas üldiselt kõige sagedamini esinevaid tabeleid või siis tabeleid, mis esinevad umbes sama tihedalt, kui ülejäänud etteantud tabelid.

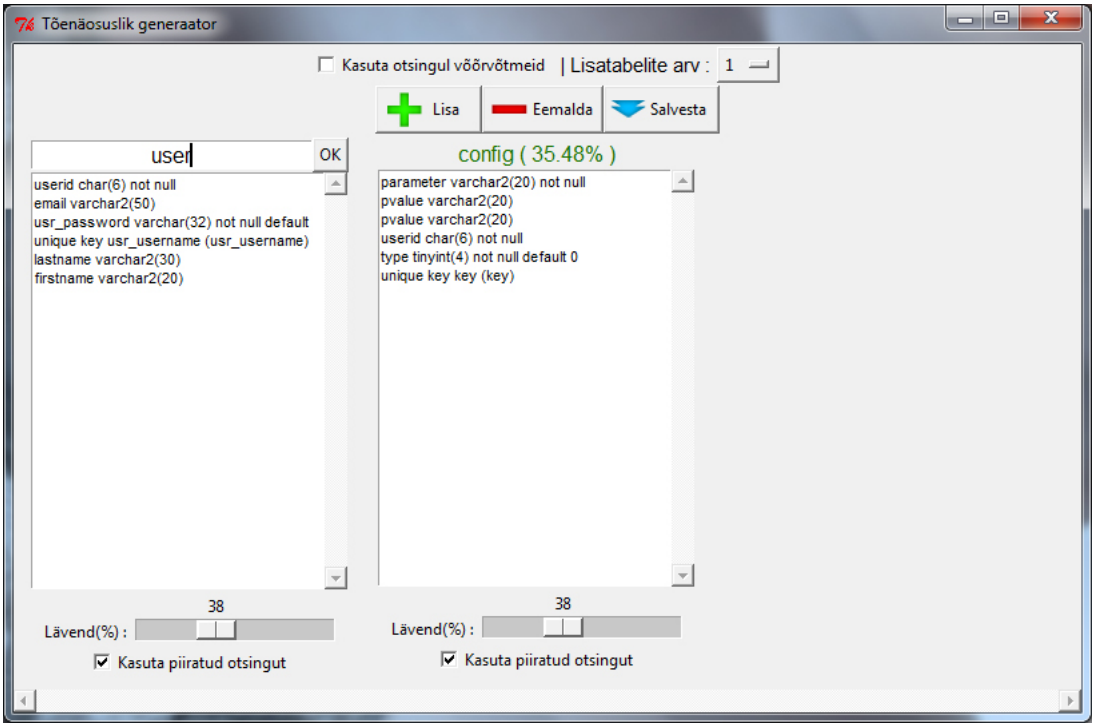
Eelmist mõtet saaks jätkata lisades tõenäoliste tabelite või veergude otsingule ka ülemise lävendi või piiri, mille ülemisel loetakse objekt liiga populaarseks. Selliste liigesindatud objektide ignoreerimine võimaldaks sundida algoritmi leidma lahendit haruldasemate skeemade hulgast. Seega kui kasutaja saab aru, et soovitakse luua midagi võrdlemisi unikaalset, oleks võimalus otsida vastuseid ka haruldase ehitusega skeemade hulgast.

Mõistlik võiks olla ka parameetri lisamine, mis võimaldaks kallutada algoritmi tööd lihtsa otsingu ja tõenäosusliku otsingu vahel. Kui teadmusbaas kasvab piisavalt suureks võime kalduda rohkem parima vaste leidmisele - tagastatakse üks teadmusbaasis reaalselst eksisteeriv skeema. Kui aga soovitakse üldistavamat vastus, kaldume tõenäosusliku lähenemise poole - komplekteerime tõenäosusliku info põhjal uue skeema. Antud ettepanek oleks võimalik realiseerida näiteks arvestades skeemas leiduvate tabelite arvu. Kui piirata väljapakatud algoritmi (Peatükk 2.5) vaadeldes otsingul ainult skeemasid millel on kindel arv tabeleid, saaksime oluliselt kitsendada otsingut ning seega leida tõenäolisi lahendeid ainult kitsas alamhulgas. Kuidas selline sobilik tabelite arvuline kitsendus leida on aga omaette küsimus. See võiks olla määratud nii kasutaja poolt kui ka algoritmiliselt (nt vastuste otsingul kasutatakse ainult skeemasid, kus leidub poolteist korda rohkem tabeleid

kui kasutajapoolt tabeleid ette anti).

3.5 Kasutajaliides

Järgnevalt kirjeldame põgusalt väljapakutud algoritmile loodud graafilist kasutajaliidest. Antud rakendus kasutab ainult Python 3.2 baasteeke ja peaks seega käivituma ilma lisateeke vajamata. Graafilise kasutajaliidese avamiseks käivitage fail *gui.py*. Programmi töö käivitumisel peaks avanema aken sarnane joonisel 3.6 nähtavaga. Graafilise kasutajaliidese võib üldiselt jagada kaheks - üldised seaded ja tabelite info.



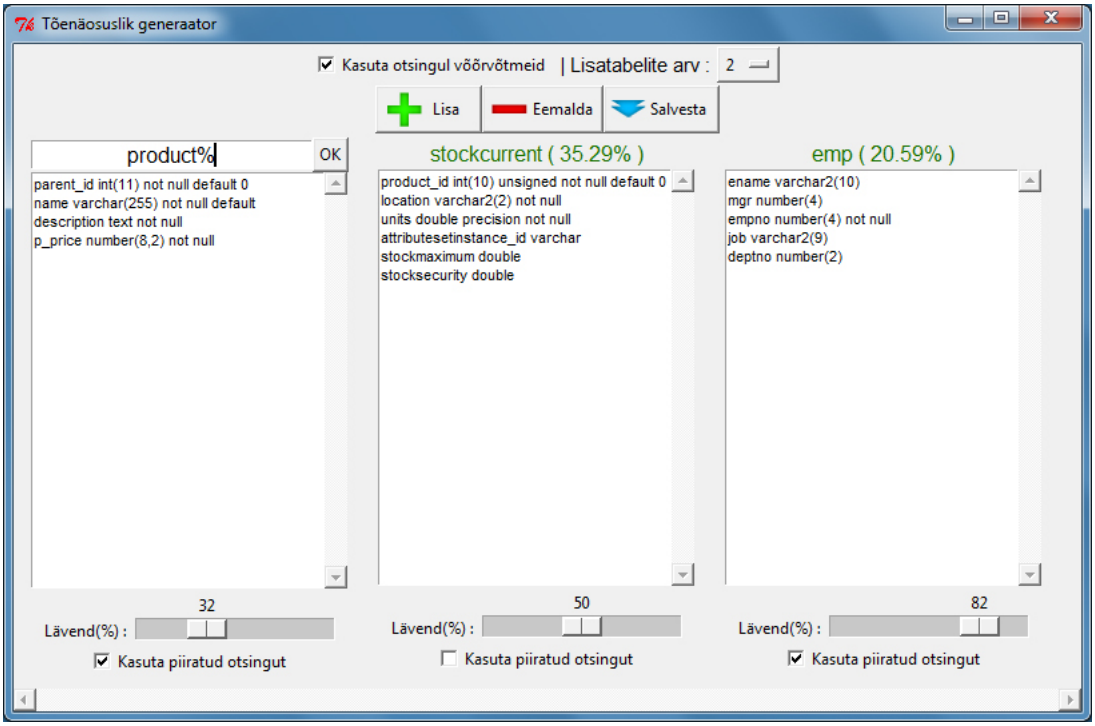
Joonis 3.6: Automaatse andmebaasi mudelite genereerimise algoritmi graafiline kasutajaliides. Näide 1.

Üldised seaded (joonis 3.6) asuvad akna ülemises servas ning võimaldavad juhtida algoritmi üldist tööd. Märkeruut “Kasuta otsingul võõrvõtmeid” muudab lisatablite otsingul tõenäolistemaks need tabelid, millele viidatakse või mis viitavad etteantud tabelitele ka võõrvõtmete abil. Rippmenüüst “Lisatabelid” saab määrata soovitud lisatabelit arvu. Nupu “Lisa” abil saab lisada ning nupu ”Kustuta” abil kustutada etteantud tabeleid. Nupp “Salvesta” salvestab hetkel programmis nähtava skeema tabelite loomise SQL laused faili. Kasutajal tuleb seejuures määrata faili nimi ning salvestamise asukoht.

Tabelite info osas (joonis 3.6) kuvatakse nii etteantud kui ka väljapakutud lisatabelid koos nendes leiduvate veergudega. Peale “Lisa” nupu vajutamist ilmub

nähtavale uus tabel millele tuleb anda nimi. Peale nime sisestamist ja "OK" nupu (või klaviatuuril "Enter") vajutamist alustab algoritm tööd ning leiab kõige sagedamini esinevad veerud. Veerude nimistus näidataks ära lisaks veeru nimele ka andmetüüp. Tabelite all asuvat "Lävendi" kerimisriba saab kasutada määramaks kui palju veerge tabelisse soovitakse. Mida madalam on tõenäosuslik lävend, seda rohkem veerge lisatakse. Märkeruut "Kasuta piirantud otsingut" tabeli all sunnib võimalike veergude otsingut piirama ainult skeemadele kus asuvad kõik etteantudning lisatabelid. Lisatabelite puhul näidataks kasutajale tabelinime kõrval konkreetse tabeli esinemise tõenäosust.

Joonisel 3.7 on näha ekraanipilt rakendusest olukorras, kus ette on antud tabelinimi "product%" ning lastud lisada kaks lisatabelit. Tulemusena genereeritud täielik SQL definitsioonifail on nähtav Lisas 4.



Joonis 3.7: Automaatse andmebaasi mudelite genereerimise algoritmi graafiline kasutajaliides. Näide 2.

3.6 Tehtud eeldused

Esiteks eeldame, et algoritmi kasutaja omab ettekujutust, millist andmemudelit soovitakse. Ta peab olema suuteline identifitseerima vähemalt mõned olulisemad andmeobjektid, mida andmebaasis oleks tarvis salvestada. Esitletud algoritm pakub küll välja tõenäoliselt sobiliku skeema, kuid täpsema lõppvastuseni suunamine jääb kasutaja hooleks.

Samuti peab algoritmi teadmubaas olema piisavalt suur, et sisaldada erinevate ainevaldkonna infosüsteemide andmemudeleid. Mida suurem on teadmusbaas, seda mitmekesisemaid ning paremaid tulemusi suudab algoritm genereerida. Andmete rohkus võimaldaks ka kalduda rohkem täpsete vastete otsimisele.

Kuna enamus teadmusbaasis leiduvatest skeemadest kasutab inglise keelseid tabelite ning veergude nimesid, siis tuleb ka algoritmi kasutamisel etteantud tabelite nimed sisestada inglise keeles. Seega kasutaja peab andmeobjektide nimetamisel kasutama sama keelseid termineid nagu seda tehakse näidisskeemades.

Kasutamismugavuse seisukohast eeldame, et algoritmi kasutatakse peatükis 3.5 kirjeldatud graafilise kasutajaliidese vahendusel. Kuigi algoritm on realiseeritud ka püsiprogrammeeritud kujul (failis *implementation.py*) ning poolautomaatsel kujul (failides *comparison.py* ja *autoimplementation.py*) tuleb algoritmi jooksva suunamise poolt saadav eelis välja just graafilise kasutajaliidesega.

Viimaseks eeldame, et näidisandmete töötlemine ning salvestamine toimub harva ning enne seda kui kasutaja reaalselt antud algoritmi skeemade genereerimiseks kasutama hakkab. Seoses sellega ei ole skeemade töötlemise ning salvestamisele seatud erilisi ajapiiranguid. Enamikele teadmusbaasi loodavate tabeli veergudele on lisatud indeksid ning skeemade süntaksi analüüs toimub regulaaravaldiste abil.

Peatükk 4

Testimine ja tulemused

4.1 Testimise üldine kontseptsioon

Väljapakutud algoritmi töö headusele hinnangu andmiseks oleks tarvis seda võrrelda alternatiivsete lahendustega. Lisaks peatükis 2.2 mainitud otsingule kasutame ka lihtsustatud versioone meiepoolt väljapakutud algoritmist. Nimekiri algoritmidest mida testimiseks kasutasime:

A0 Esimese vaste otsing. Etteantud tabelinimede põhjal sooritatakse treeningandmebaasi päring. Päringu vastusena tagastatakse kõik skeemad kus leiduvad etteantud tabelid. Vastuste seast valitakse üks juhuslik skeema ning tagastatakse see. Kõige lihtsam lahendus püstitatud probleemile ja kasutame baaslahendusena võrdlemisel väljapakutud algoritmite erinevate versioonidega.

A1 Väljapakutud algoritm (tõenäosuslik otsing) ilma lisatabeliteta.

A2 Väljapakutud algoritm (tõenäosuslik otsing) koos lisatabelitega. Kasutatakse võõrvõtmete informatsiooni tõenäoliste lisatabelite leidmisel.

A3 Väljapakutud algoritm (tõenäosuslik otsing) koos lisatabelitega. Kasutatakse võõrvõtmete informatsiooni tõenäoliste lisatabelite leidmisel. Juhul kui ei leidu kõigi etteantud tabelitega skeemat, siis teostatakse sõltumatu tabelite otsing.

Eelmainitud meetodite testimiseks kasutasime traditsioonilist andmete (skeemade) jagamist treening- ning testhulgaks. Treehinghulka paigutati 90% ning testhulka 10% näidisandmetest. Siinkohal juhime tähelepanu asjaolule, et väljapakutud algoritmi normaaltingimuslik realisatsioon (nt graafilise kasutajaliidesega) kasutab treeninghulgana ehk teadmusbaasina 100% andmetest. Seega algoritmide testimiseks loodi eraldi andmebaasid. Testimise idee seisneb selles, et testandmete hulgast valitakse välja skeema, mis hakkab esindama n-ö soovitud tulemust. Seejärel valitakse antud skeemast juhuslikult teatud kogus tabelinimesid ning kasu-

tatakse neid eelmainitud nelja meetodi sisendina. Seega algoritmidele antakse ette alamhulk soovitud skeema tabelinimedest ning lastakse neil välja pakkuda vastus. Vastuse leidmiseks kasutatakse andmeid treeninghulgast, mis esindab siis hetkel algoritmide teadmusbaasi. Algoritmi üldise töökvaliteedi hindamiseks võrdleme lahendina saadud ning soovitud skeema sarnasust. Mida sarnasemad on etteantud ning vastuse skeema seda parema tulemuse on algoritm saavutanud. Vältimaks tõenäoliselt haruldasi (liiga häid või halbu) tabelinimede valikuid, korratakse tabelinimede valikut ning algoritmide tööd kümme korda, misjärel võetakse sarnasuse hinnangutest aritmeetiline keskmine. Kuna soovitud tulemit esindav skeema võib juhtuda olema täiesti või osaliselt unikaalne võrreldes treeningandmetega, siis sai realiseeritud järgnev lahendus: kui etteantud tabelnimede põhjal ei suuda algoritm ühtegi sobiliku skeemat treeningandmete hulgast leida, siis vähendatakse etteantud tabelite loendit ühe võrra ning korratakse otsingu protseduuri. Sellisel viisil etteantud tabelinimede eemaldamine laiendab otsinguid ning võimaldab ka halvemate vastuste leidmist. Algoritmide töö lõppeb, kui leitakse sobiv vastus või etteantud tabelinimed loend on tühjenenud – sellisel juhul vastus puudub.

4.2 Sarnasuse hinnangu arvutamine

Esialgne plaan oli kasutada skeemade sarnasuse hindamiseks mõnda olemas olevat lahendust. Nimelt on skeemade kokkusobitamise või sarnasuse hindamisese valdkonnas tehtud väga palju uurimustööd. Erinevate infosüsteemide ühildamisel on kasulik kui andmebaaside ühendamine toimuks automaatselt, kuna käsitsi seoste leidmine on väga aeganõudev ning rutiine tegevus. Skeema-sobitus algoritmidest on tuntumad COMA++, CUPID, AUTOPLEX, ASID jne. Kuigi paljud väljapakutud lahendused keskenduvad rohkem XML ja OWL andmeskeemadele ning tagastavad ainult skeemade kokkusobitamise plaani, siis mõned töötavad ka relatsiooniliste mudelite peal ning lisaks arvutavad välja skeemade objektiivse sarnasuse hinnangu. Just viimastest algoritmidest (nt COMA++) olimegi huvitatud. Kuid probleemseks osutus nimetatud algoritmide realisatsioonide hankimine. Teadusartiklitest saab küll teada algoritmide tööpõhimõtte, kuid konkreetsed implementatsioonid puuduvad. Kuna enamus tänapäevaseid lahendusi on võrdlemisi mahukad ning keerulised (kasutatakse nii masinõpet kui tehisintellekti tehnikaid), siis otsustasime ise neid mitte realiseerima hakata. Katsed võtta algoritmide loojatega ühendust ning paluda prototüübi koodi jäid tulemusteta. Seega olime sunnitud ise välja mõtlema sarnasuse hinnangu funktsiooni.

Algoritmi tööle hinnangu andmisel tekkivad sarnased küsimused nagu skeemade kokkusobitamisel. Nimelt milliseid statistilisi näitajaid tulemuste kvaliteedi hindamiseks vaadelda. Paljud skeemade kokkusobitamise algoritmid lähtuvad vastuste otsimisel hinnangufunktsioonidest, mis eelistavad täpsust. Reaalseses kasutamistingimustes aga selgub, et selline lähenemine ei ole alati kõige mõistlikum. Saagist ja täpsust puudutavat probleemi on varemgi uuritud [Gal07], [DBC08]. Kujutame

ette, et meil on kaks skeemat milles mõlemas leidub 100 elementi (seega kui vaadelda ainult 1:1 seoseid on erinevaid kokkusobitamiste võimalusi 10000). Oletame, et 25 seost on neist relevantsed. Kui meie algoritm avastab 10 seost ja saavutab täpsuse 100% siis tuleks kasutajal hiljem käsitsi leida 15 puuduvat seost 81000 võimaliku variandi hulgast. Samas kui mõni teine algoritm leiaks 300 seost ja saavutaks saagise 100% siis tuleks kasutajal hiljem käsitsi eemaldada 275 seost võimaliku 300 seose hulgast. Seega kui arvestada hilisemat käsitsi tehtava lisatöö hulka, annavad suure saagisega algoritmid paremaid tulemusi. Lisaks on inimese jaoks seoste valideerimine lihtsam tegevus kui ülisuure otsinguruumi läbi vaatamine.

Algoritmi vastusele hinnangu andmiseks kasutasime Jaccardi koefitsientide [Jac12] võrdlemist. Üldist võimet leida õigeid tabeleid sisaldavaid skeemasid hindame järgnevalt. Oletame, et hulk *Soovitud tabelid* sisaldab tabelinimesid, mis pärinevad soovitud tulemust esindava testandmete skeemast ning hulk *Vastuse tabelid* sisaldab tabelinimesid, mis pärinevad algoritmi poolt vastusena tagastatud skeemast. Sellisel juhul saame tabelinimede sarnasuse Jaccardi koefitsiendi leidmiseks valemi:

$$J_{tabelid} = \frac{Soovitud\ tabelid\ \cap Vastuse\ tabelid}{Soovitud\ tabelid\ \cup Vastuse\ tabelid} \tag{4.1}$$

Üldist skeemasse salvestava informatsiooni täpsuse hindamiseks kasutasime modifitseeritud kujul Jaccardi koefitsiendi leidmist. Nimelt leiame iga ühise tabeli (element hulgast *Soovitud tabelid* $\cap$ *Vastuse tabelid*) puhul selle tabeli veergude sarnasuse koeftsiendi. Oletame, et hulk *Soovitud veerud* sisaldab veerunimesid, mis pärinevad soovitud tulemust esitava testandemete skeema tabelist X ning hulk *Vastuse veerud* sisaldab veerunimesid, mis pärinevad algoritmi poolt vastusena tagastatud skeema tabelist X. Sellisel juhul saame veerunimede sarnause Jaccardi koefitsendi leidmiseks valemi:

$$J_{Xveerud} = \frac{Soovitud\ veerud\ \cap Vastuse\ veerud}{Soovitud\ veerud\ \cup Vastuse\ veerud} \tag{4.2}$$

Üldise *veergude sarnasuse* koefitsiendi leidmiseks võtame kõikidest tabeli veerusarnasustest aritmeetilise keskmise. *Veergude sarnasus* väljendab seega tabelite veergude keskmist sarnasust. Üldise andmete salvestamise informatsiooni täpsuse hinnangu valemi saab esitada nüüd kujul:

$$J_{informatsioon} = \frac{(Soovitud\ tabelid\ \cap Vastuse\ tabelid) \cdot Veergude\ sarnasus}{Soovitud\ tabelid\ \cup Vastuse\ tabelid} \tag{4.3}$$

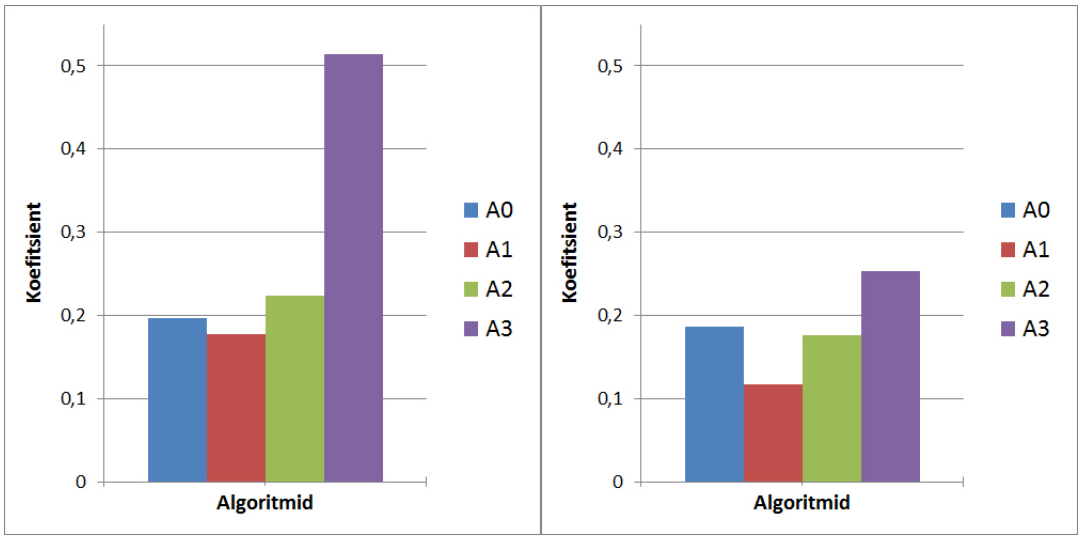
Kõik koefitsiendid asuvad vahemikus [0..1], kusjuures 0 tähendab täielikult mittekattuvaid skeemasid ning 1 täielikult identseid skeemasid.

4.3 Testimise tulemused

Kokku oli näidisandmeteks 518 skeemat, millest 10% kasutati testandmetena. Seega jäi testimiseks 52 skeemat. Iga skeema kohta arvutasime peatükis 4.2 esitatud

koefitsiendid. Tulemustest parema ülevaate saamiseks esitame graafiliselt siinkohal ainult nende aritmeetilised keskmised. Kasutatud algoritmid on märgendatud vastavalt peatüki 4.1 kirjeldustele.

Esiteks vaatleme algoritmide võimet leida võimalikult sarnaste tabelitega skeemasid. Selleks arvutame valemi (4.1) põhjal tabelite sarnasuse koefitsiendid. Joonisel 4.1 näeme nelja algoritmi suutlikust erinevate *informatsiooni koguste* juures. Informatsiooni koguseks nimetame parameetrit, mis näitab protsentuaalselt kui suurt osa soovitud tulemust esindava skeema tabelinimedest kasutatakse algoritmidele sisendandmetena.



a) Informatsiooni kogus 50%

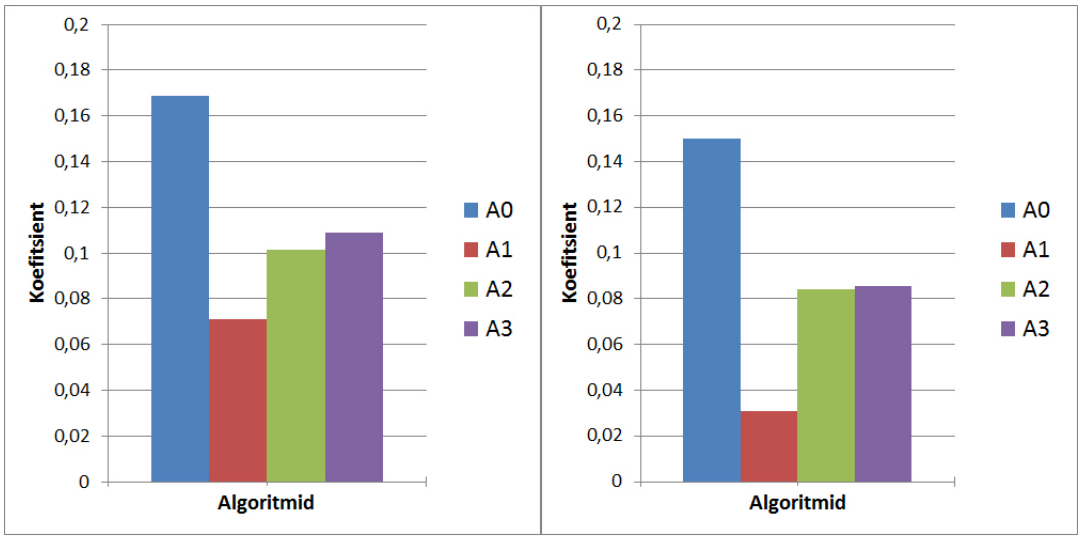
b) Informatsiooni kogus 20%

Joonis 4.1: Tabelite sarnasuse koefitsiendid

Nagu graafikult näha on algoritmide tulemused vähese informatsiooni korral tunduvalt võrdsemad. See on ilmselt märk sellest, et skeemad on üldjoontes suhteliselt unikaalsed. Kui algoritmidele antakse ette väga vähe tabelinimesid, siis tõenäosuslikud vastused (A2, A3) ei suuda esimese vaste otsingut (A0) enam märkimisväärselt ületada. Algoritm A1 sooritab alati halvemini kui võrdlusbaasina kasutatud A0, kuna ei kasutata lisatabelite otsingut ning piirdutakse ainult etteantud tabelite tagastamisega.

Huvitavam tulemuse saime aga üldist informatsiooni salvestamise täpsust arvutades (valem 4.3). Nagu jooniselt 4.2 näeme suudab algoritm A0 sooritada teistest märkimisväärselt paremini. See esialgu üllatav tulemus sundis tegema lisakatsetusi leidmaks selle üllatusliku tulemuse põhjuseid. Meiepoolt väljapakutud algoritmi realisatsiooni erinevad versioonid (algoritmid A1, A2, A3) leidsid iga tabeli kohta 10 kõige tõenäolisemat veergu[1] ning tagastasid need. Oletades, et sellisel viisil konstantselt etteantud veergude arv sooritab loodetust halvemini, kuna paljudes

[1]Väljapakutud algoritmi tugevus seisneb tegelikult dialoogis kasutajaga ja seega on algoritmi normaaltingimuslikus realisatsioonis ettenähtud, et kasutaja määrab soovitud veergude arvu ise.



a) Informatsiooni kogus 50%

b) Informatsiooni kogus 20%

Joonis 4.2: Informatsiooni sarnasuse koefitsiendid

tabelites on veerge kas rohkem või vähem, muutsime enda väljapakutud algoritmi automaatset realisatsiooni. Algoritmid üritasid leida iga tabeli juurde täpselt sama palju veerge, kui oli soovitud tulemust esindavas skeemas. Selgus aga, et selline lahenduse korral sooritasid algoritmid veel halvemini kui fikseeritud 10 veeruga tabelite leidmine. Samuti andis rohkem kui 10 veeru kasutamine halvemaid tulemusi.

Seega saame teha ainult järelduse, et väljapakutud informatsiooni sarnasuse hinnang (valem 4.3) on ehk isegi liiga andestav suurestes kogustes veergude lisamisele senikaua, kuni selles hulgas leidub ka mingi hulk õigeid veerge. Samas kui tuletame meelde peatükis 4.2 käsitletud probleemi saagise ja täpsuse vahel, siis ehk ei olegi saadud tulemuse kallutatus eelistama rohkemaid veerge väga ekslik. Valede veergude hilisem käsitsi eemaldamine on lihtsam kui uute lisamine.

Kuid küsimus, et miks esimese vaste otsing (algoritm A0) suutis informatsiooni salvestamise sarnasuse mõistes sooritada märkimisväärselt paremini kui tõenäosuslikud otsingud, sundis eraldi uurima algoritmide sisendeid ja väljundeid. Nagu lähemal vaatlemisel selgus oli A0 edu suuresti tingitud probleemidest treening- ja testandmetega. Ühest küljest andmeobjektid, mida skeemades salvestatakse, on tihti erinevatest rakendusvaldkondadest ja seega ka täiesti unikaalsed. Sellis-te unikaalsete skeemade genereerimisel on kõik vaadeldavad algoritmid suuresti sarnastes probleemides (vastusena ei suudeta midagi täiesti uut genereerida). Kuigi algoritmid A1, A2 ja A3 sooritavad antud olukorras natuke paremini kui A0, on erinevus siiski tühine. Ja teisest küljest on paljudest skeemadest dublikaadid - täiesti identsed skeemad või siis natuke uuem või vanem versioon samadest andmemudelitest. Seega esineb üsna sagedasti olukord, kus testandmete hulgast valitud soovitud tulemust esindav skeema leidub sarnasel kujul ka treeningand-

mete hulgas. Kuna tõenäosuslikud algoritmid kasutavad vastuse koostamiseks ka teisi sarnaseid skeemasid, siis ongi tulemuseks olukord, kus esimese vaste otsing (algoritm A0) tagastab sageli tunduvalt parema vastuse. Selliste täpsete vastuste leidmine treeningandmete hulgast kallutabki sarnasusekoefitsiendi A0 kasuks.

Kokkuvõtvalt võib öelda, et väljapakutud algoritm sooritab võrdlemisi stabiilselt nii seni nägemata kui ka treeningandmete hulgas olevate andmete peal. Kuigi tegelikult on treeningandmeid väga vähe ja skeemad on võrdlemisi unikaalsed, siis andmemudelite kohatine dubleeritus loob algoritmidele illusiooni treeningandmete paljususest. Selle tagajärjel sooritab esimese vaste otsing (A0) palju paremini kui ta peaks. Kui antud kontekstis A0 sobitab vastuseid üle, siis väljapakutud algoritm ehk jällegi alasobitab neid.

Algoritmide võrdlemine on realiseeritud failis *comparison.py*. Väljapakutud algoritm on realiseeritud automatiseeritud kujul failis *autoimplementation.py*.

4.4 Olukorrad, kus algoritm ei pruugi töötada

Ebastandardsed skeema SQL definitsioonid võivad põhjustada selle valesti tõlgendamist. Regulaaravaldised suudavad küll ignoreerida näitakes HTML märgendit, kuid meelvaldne sulgude kasutamine on lubamatu. Kui süntaksi ülesehitust ei suudeta tuvastada ignoreeritakse antud definitsiooni. Kuid probleem tekkib juhul kui analüüs õnnestub ainult osaliselt - definitsiooni struktuur suudetakse tõlgendada osaliselt. Sellisel juhul kogutakse skeemast n-ö korrupeerunud informatsiooni (objektide nimed ning parameetrid sisaldavad väärat informatsiooni). Tulemusena võidakse teadmusbaasi lisada andmeid, mis ei oma sisulist väärtust ning pigem segavad lahendi leidmist mõjudes mürana.

Täiesti seninägemata ehk teadmusbaasis mitte esinevate skeemade genereerimisel annab algoritm üsna kesiseid tulemusi, kuna sellisel juhul ei osata leida võimalike puuduvaid lisatabeleid. Peatükis 3.4 esitatud ettepanekute abil oleks võimalik antud probleemi leevendada. Sarnast olukorda põhjustab ka treeningandmete vähesus. Kui teadmusbaas on väga väike, siis on tinglikult kõik skeemad seni nägematud ning unikaalsed.

Kuna algoritmi töö sõltub tugevalt tabelinimedest, siis ollakse ka tundlik nimede variatsioonidele. Näiteks võib ühe andmeobjekti nimetus mitmuses anda erinevaid vastuseid võrreldes selle ainsuse vormiga (nt “user” vs “users”). Samuti on algoritm tundlik kirjavigadele ning olukordadele, kus levinud andmeobjekte üritakse nimetada ebatraditsiooniliste nimedega (nt kasutajainfo salvestamiseks kasutada tabelinime “inimesed”).

Algoritmi tööprotsessi ajal tehtavad muudatused teadmusbaasi võivad tingida ebajärjepidava vastuse (vastus ei vasta andmetele mis asusid teadmusbaasis enne ega ka peale muudatusi). Kuna algoritm komplekteerib lõppvastuse paljudest

alamosadest ning teostab seejuures mitmeid päringuid andmebaasi on võimalik olukord, kus andmebaas muudatused algoritmi töö ajal rikuvad lõppvastuse.

Kokkuvõte

Antud magistritöö eesmärk oli uurida võimalusi andmebaasi mudelite automaatseks genereerimiseks ning pakkuda välja ka võimalik lahendus antud probleemile. Kuna andmebaaside loomine on tänapäeval infosüsteemide lahutamatu osa ning skeemad on vähemalt osaliselt tihti sarnased, siis oleks mõistlik antud tegevus automatiseerida.

Töös vaadeldi skeemade automaatse genereerimise olemust ning arutleti võimalike lahendusmeetodite üle. Antud probleemi lahendamiseks esitleti ühte konkreetset tõenäosuslikul lähenemisel põhinevat algoritmi. Lisaks abstraktsele algoritmi töökirjeldusele toodi välja ka üks võimalik realisatsioon, koos seletuste ning põhjendustega. Algoritm realiseeriti programmeerimiskeeles Python ning loodi ka graafiline kasutajaliides. Samuti arutleti alternatiivsete lahendusmeetodite üle - vaadeldes nii teistsuguseid lähenemisi kui ka võimalusi väljapakutud algoritmi parendamiseks. Töö viimases osas anti hinnang väljapakutud algoritmi tulemustele, võrreldes seda mõnede alternatiivsete lahendustega.

Töö käigus selgus, et kuigi tõenäosuslike vastuste genereerimine annab rahuldavaid tulemusi, on sellel siiski ka puudujääke. Õnneks on enamus tekkinud probleeme võimalik lahendada kasutades erinevaid masinõppe lähenemisi. Kõige enam valmistasid raskusi SQL skeema definitsioonifailide süntaksi analüüsimine ning skeemade genereerimise algoritmi üldkonseptsiooni väljamõtlemine. Huvitavaks tegi antud töö asjaolu, et suuresti puudus varasem uurimustöö, millele toetuda ning paljudele probleemidele tuli ise jooksvalt lahendusi leida.

Kokkuvõtvalt arvan, et tööle püstitatud eesmärgid said täidetud ning skeemade automaatne genereerimine esmasel kujul ka realiseeritud. Väljapakutud lahendus pole aga kindlasti mitte ammendav ning seda oleks võimalik tulevikus edasi arendada.

Automatic data-driven generation of database schemas

Master's thesis (30 ECTS)

Joel Edenberg

Summary

The goal of this thesis was to study the possibilities for automatically generating database schemas and to implement a proof-of-concept. All information systems contain some sort of means to store information - often a relational database is used. In order to store the adequate data, we need a suitable database schema. As it turns out similar software applications also share at least partly similar database schemas. So it should be theoretically feasible to generate schemas, or parts of them, automatically.

In the first chapters of the thesis we discuss some of the general possible approaches. We proposed a novel algorithm for solving the task of generating schemas. In order to find the most common or most probable solution the proposed algorithm uses a probabilistic model. The algorithm is given a partial list of table names, desired in the resulting schema. These table names represent the data objects user wants to store. In essence the given table names tell the algorithm what data needs to be saved and leaves it up to the program to compose the entire solution.

We created one possible implementation of the proposed algorithm (written in Python). Our proposed prototype takes heavy usage of dialogue-like interaction with user. A graphical user interface was also made in order to enhance the working experience and ease the tuning of the algorithm. As the user is the only one who is fully knowledgeable of the requirements, we left several configuration parameters up for fine tuning by user. In addition to given table names user can also determine how many additional tables would be needed (tables that algorithm could find also relevant and append to the solution), should we use database foreign keys too for finding relative additional tables, how many columns do we want in each table and how specific should the column definitions be to the current schema.

Next we discussed alternatives solutions, potential improvements and possibilities for future research. In the last part of the thesis we experimentally assessed the perfomance of our algorithm and compared several variations of it. We introduced

a novel similarity measure between two schemas in order to estimate the quality of the answers. Due to some specifics of the chosen knowledge base (database containing data about schema examples) our algorithm turned out not to be better than a naive first-match search. However, we believe that in practice using the probabilistic algorithm yields to better results.

Kirjandus

[Ble12] David Blei. Foundations of probabilistic modeling. `http://www.cs.princeton.edu/courses/archive/fall10/cos513/`, May 2012.

[BM02] Jacob Berlin and Amihai Motro. Database schema matching using machine learning with feature selection. In *In Proceedings of the Conf. on Advanced Information Systems Engineering (CAiSE*, pages 452–466, 2002.

[DBC08] Fabien Duchateau, Zohra Bellahsene, and Remi Coletta. A flexible approach for planning schema matching algorithms. In *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems:*, OTM '08, pages 249–264, Berlin, Heidelberg, 2008. Springer-Verlag.

[DM86] Gerald Dejong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1:145–176, 1986. 10.1007/BF00114116.

[DMR03] Hong-Hai Do, Sergey Melnik, and Erhard Rahm. Comparison of schema matching evaluations. In Akmal Chaudhri, Mario Jeckle, Erhard Rahm, and Rainer Unland, editors, *Web, Web-Services, and Database Systems*, volume 2593 of *Lecture Notes in Computer Science*, pages 221–237. Springer Berlin / Heidelberg, 2003.

[DR02] Hong-Hai Do and Erhard Rahm. Coma: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 610–621. VLDB Endowment, 2002.

[Fou12] The Python Software Foundation. Python documentation. `http://www.python.org/doc/`, May 2012.

[Gal07] Avigdor Gal. The generation y of xml schema matching panel description. In Denilson Barbosa, Angela Bonifati, Zohra Bellahšne, Ela Hunt, and Rainer Unland, editors, *Database and XMLTechnologies*, volume 4704 of *Lecture Notes in Computer Science*, pages 137–139. Springer Berlin / Heidelberg, 2007.

[GL09] Jan Goyvaerts and Steven Levithan. *Regular Expressions Cookbook - Detailed Solutions in Eight Programming Languages.* O'Reilly, 2009.

[Hwa12] Hwaci. Sqlite documentation. `http://www.sqlite.org/docs.html`, May 2012.

[Jac12] Paul Jaccard. The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):pp. 37–50, 1912.

[Mit97] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, New York, 2 edition, 1997.

[Mit98] Isi Mitrani. *Probabilistic modelling.* Cambridge University Press, New York, NY, USA, 1998.

[OD08] David L. Olson and Dursun Delen. *Advanced Data Mining Techniques.* Springer Publishing Company, Incorporated, 1st edition, 2008.

[Ris01] I Rish. An empirical study of the naive bayes classifier. *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, 3(22):41–46, 2001.

[VR79] C.J. Van Rijsbergen. *Information retrieval.* Butterworths, 1979.

[WZ03] Hua Wu and Ming Zhou. Optimizing synonym extraction using monolingual and bilingual resources. In *Proceedings of the second international workshop on Paraphrasing - Volume 16*, PARAPHRASE '03, pages 72–79, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

Lisad

Lisa 1. Terminoloogia

Andmemudel Andmebaasis andmete struktureerimise viis, millega üritatakse kirjeldada reaalse maailma andmeobjekte.

Skeema on formaalses keeles kirjeldatud andmeebaasi täielik struktuur. See on n-ö detailne plaan andmebaasis leiduva informatsiooni ehituse kohta.

Teadmusbaas Andmete kogum, mis esindab kogu informatsiooni, mis on algoritmile kättesaadav. Antud kontekstis koosneb teadmusbaas skeemade ehitust puudutavast informatsioonist.

Näidisandmed Reaalselt kasutuses olevate infosüsteemide skeema definitsioonifailid.

Treeningandmed Näidisandmed mida kasutatakse teadmusbaasi loomiseks.

Treeninghulk Osa näidisandmetest, mida kasutatakse algoritmi töökvaliteedi testimisel selle teadmusbaasina. Tavaliselt ligikaudu 80% näidisandmetest.

Testhulk Osa näidisandmetest, mida kasutatakse algoritmi töökvaliteedi testimisel selle testbaasina. Testhulgas leiduvaid andmeid kasutatakse ideaalse vastusena ning neid võrreldakse algoritmi poolt tagastatud tulemustele.

SQL (*Structured Query Language*). Programmeerimiskeel, mis on loodud andmete haldamiseks relatsioonilistes andmebaasides.

SQLite on avatud lähtekoodiga relatsioonilise andmebaasi mootor (kasutab SQL programmeerimiskeelt), mis võimaldab kompaktset, serveri- ning konfiguratsioonivaba juurdepääsu relatsioonilisele andmebaasile.

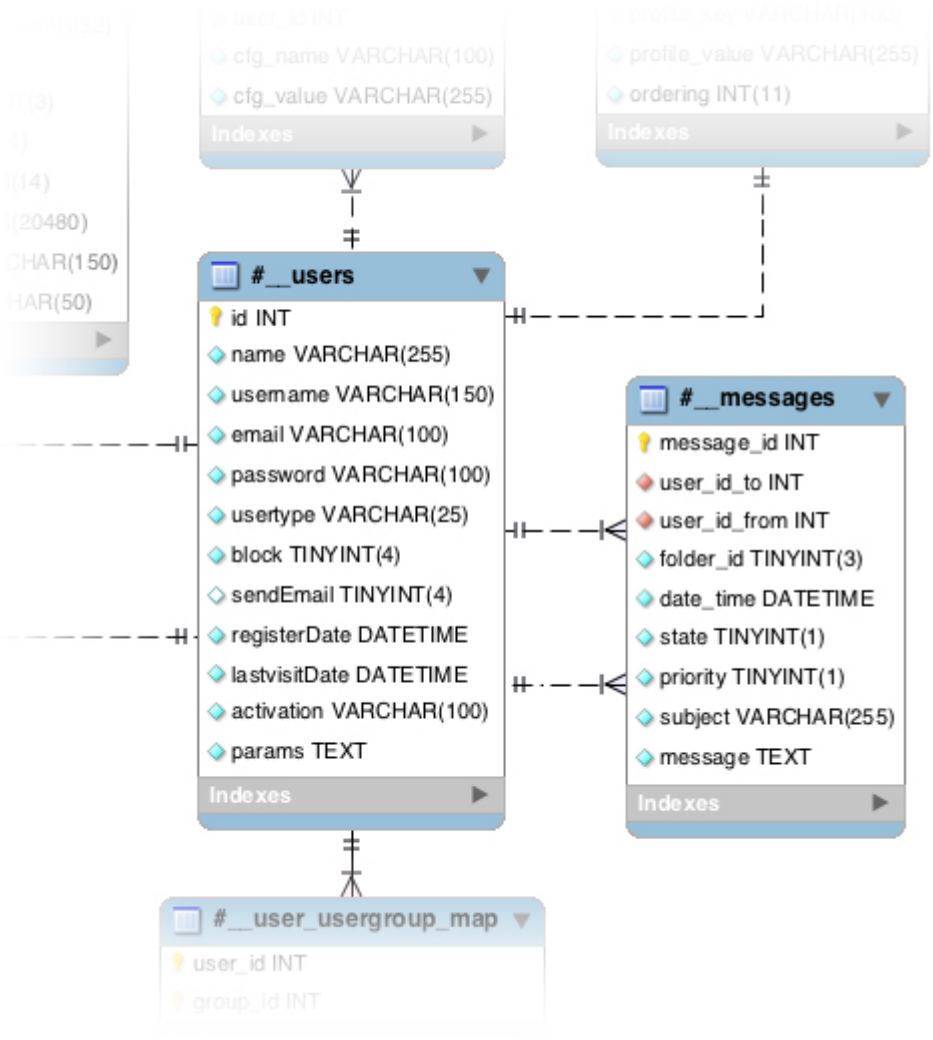
Võõrvõti (inglise keeles *foreign key*). Viitav kitsendus kahe relatsioonilise andmetabeli vahel. Kasutatakse tabelite sidumiseks andmete abil (võõrvõtmega määratakse üheselt ära millised andmed tabelites kokku sobivad).

Regulaaravaldis (inglise keeles *regular expressions*). Regulaaravaldis on sõne (tähtede ja sümbolite kombinatsioon), mis kirjeldab või langeb kokku mingi sõnede hulgaga vastavalt kindlatele süntaksireeglitele.

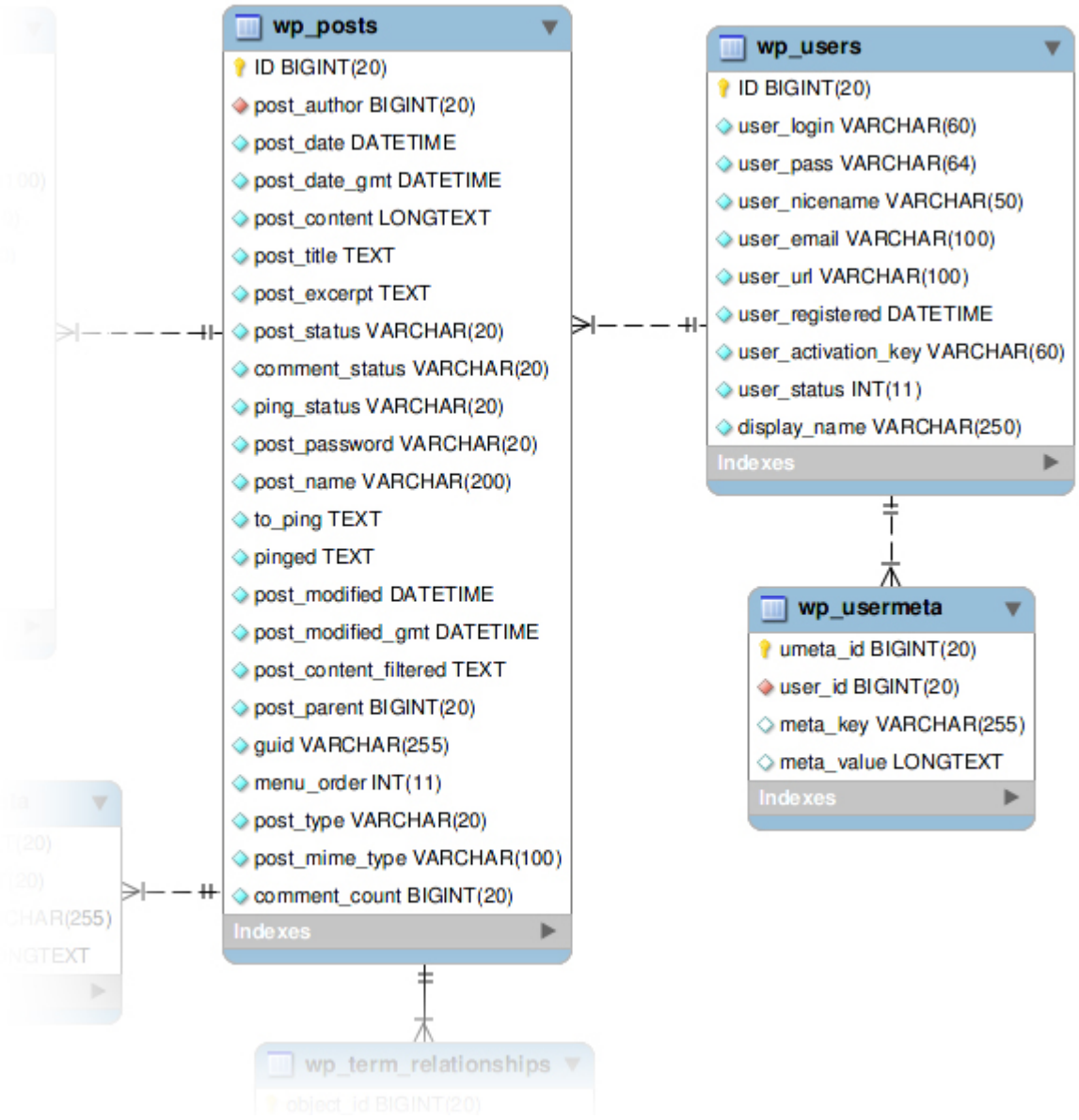
Jaccardi koefitsient Sarnasuse hinnang, mida kasutame skeemade võrdlemisel.

Tinglik tõenäosus (*conditional probability*) $P(A|B)$. Sündmuse A esinemise tõenäosus juhul kui B esineb.

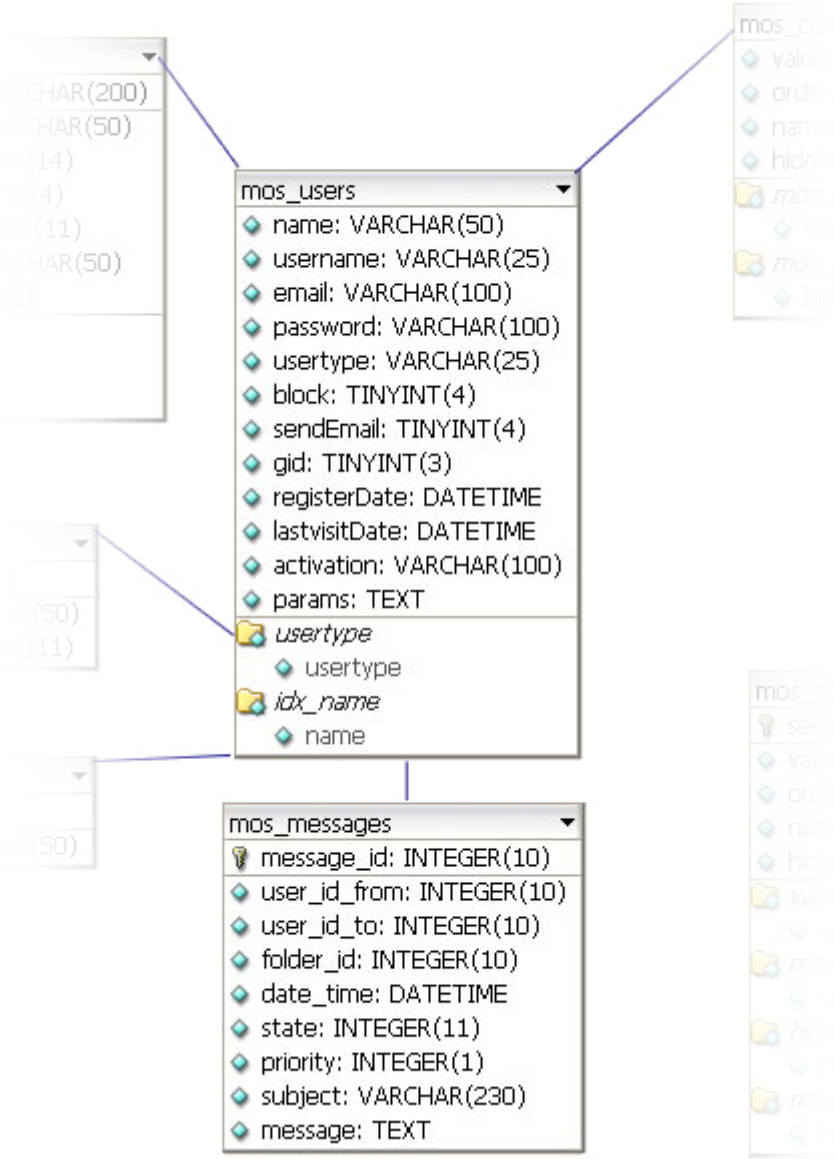
Lisa 2. Sisuhaldussüsteemide skeemad



Joonis 1: Joomla sisuhaldussüsteemi andmebaasimudeli alamosa.



Joonis 2: WordPress sisuhaldussüsteemi andmebaasimudeli alamosa.



Joonis 3: Mambo sisuhaldussüsteemi andmebaasimudeli alamosa.

Lisa 3. Skeema genereerimiseks kogutud andmed

```
Given tables to the algorithm:
['users', 'comments']

Most probable additions:
('groups', 66.67)

Most probable additions adjusted by foreign keys:
('groups', 66.67)
```

```
[ users ]
id tinyint(3) unsigned not null default 0
password varchar(100) not null default

[ comments ]
id tinyint(3) unsigned not null default 0
comment_txt text
subject varchar(64) not null default
pid number default 0 not null
comment_id int(11) not null default 0
name varchar(50) not null default
chanunavail_cid varchar(20) not null default

[ groups ] 66.67 %
id tinyint(3) unsigned not null default 0
name varchar(50) not null default
```

Lisa 4. Algoritmi väljund

```
DROP TABLE IF EXISTS product;
DROP TABLE IF EXISTS stockcurrent;
DROP TABLE IF EXISTS emp;

CREATE TABLE product (
  parent_id int(11) not null default 0,
  name varchar(255) not null default,
  description text not null,
  p_price number(8,2) not null
);
CREATE TABLE stockcurrent (
  product_id int(10) unsigned not null default 0,
  location varchar2(2) not null,
  units double precision not null,
  attributesetinstance_id varchar,
  stockmaximum double,
  stocksecurity double
);
CREATE TABLE emp (
  ename varchar2(10),
  mgr number(4),
  empno number(4) not null,
  job varchar2(9),
  deptno number(2)
);
```

Lisa 5. Materjalide CD-plaat

Tööga on kaasa pandud materjalide CD-plaat, mis sisaldab järgnevaid tulemeid:

- ***magistritoo.pdf*** - Magistritöö kirjalik osa.
- ***parsing.py*** - SQL skeemadefinitsiooni failide töötlemine ning leitud informatsiooni salvestamine SQLite andmebaasi.
- ***database.py*** - Teek skeemade andmebaasiga (SQLite) suhtlemiseks (andmebaasi loomine, andmete lisamine, päringute sooritamin).
- ***requests.py*** - Teek skeemade andmebaasi päringute sooritamiseks.
- ***implementation.py*** - Väljapakutud algoritmi realisatsioon.
- ***gui.py*** - Väljapakutud algoritmi graafiline kasutajaliides.
- ***autoimplementation.py*** - Väljapakutud algoritmi poolautomaatne realisatsioon mida kasutatakse algoritmide võrdlemisel.
- ***comparison.py*** - Algoritmide võrdlemine (Peatükk 4.3).
- ***database.db*** - SQLite andmebaas, mis sisaldab kõiki skeemasid (seda kasutab väljapakutud algoritmi realisatsioon).
- ***train.db*** - SQLite andmebaas, mis sisaldab 90% kõikidest skeemadest ja mida kasutatakse algoritmide võrdlemisel teadmusbaasina.
- ***test.db*** - SQLite andmebaas, mis sisaldab 10% kõikidest skeemadest ja mida kasutatakse algoritmide võrdlemisel testandmetena.
- **./SQL/** - Kataloog kus asuvad kõik algoritmi teadmusbaasi moodustavad SQL skeemade definitsiooni failid.

Samad materjalid on kättesaadavad ka veebiaadressilt `http://www.joel.ee/magister`