МАТЕМАТИЧКА ГИМНАЗИЈА

Прикупљање података о упису у средње школе у Србији од 2015. до 2017. године

Матурски рад

из предмета

Програмирање и програмски језици

Ученик: Лука Јовичић

Ментор: Ime mentora ovde

31. март 2018.

МАТЕМАТИЧКА ГИМНАЗИЈА

Апстракт

Матурски рад

Лука Јовичић

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Садржај

Глава 1

Увод

Када се спомену матурски радови, прва асоцијација су често теоријска излагања школског градива. Моја идеја је да овај рад има употребну вредност и ван контекста школе и понуди податке који су иначе тешко доступни, а имају својеврстан јавни значај. Подаци са уписа у средње школе су добар пример привидне транспарентности података – комплетан опус је јавно доступан, али како би се из њих извукле иоле корисне информације потребно је уложити значајан труд у њихово прикупљање и обраду, тако да једино што преостаје је ослањање на званичне извештаје који због самог опсега података никако не могу пружити комплетан увид са свих страна у ову тему која је од посебног значаја у тренутку када се константно најављују реформе образовног система.

Намера рада је да читав процес прикупљања и примера обраде спусти на „школски" ниво, тако да за његово схватање буде довољно разумевање градива које се предаје у средњим школама на смеровима за обдарене за математику. Услед обимности теме и величине самог пројекта, од теоријских основа ће бити покривено само оно што се непосредно користи у раду. Такође, фокус овог рада није на специфичним програмским језицима, већ на концепту и прецизно описаном поступку којим се до података долази, а комплетан изворни код је доступан у Додатку B.

1.1 Отворени подаци

Према дефиницији са Портала отворених података Републике Србије, отворени подаци су они који подаци који су *слободно доступни, приступачни, машински читљиви и доступни у отвореним форматима* [1]. Овакви подаци су посебно значајни када је реч о подацима који могу бити од интереса јавности. Уместо ослањања на постојеће извештаје,

отворени подаци допуштају верификацију резултата, њихове допуне и исправке од стране било кога.

1.1.1 Упис у средње школе у Србији

Иако Министартство просвете, науке и технолошког развоја Републике Србије има посебан сајт посвећен отвореним подацима (http://opendata.mpn.gov.rs), њиме није обухваћен и систем за упис у средње школе. Наиме, за потребе уписа користи се посебна платформа која се налази на http://upis.mpn.gov.rs. Овако објављени подаци се тешко могу назвати отвореним: иако су слободно доступни и приступачни, машинско читање је могуће али је нетривијално и они нису доступни у отвореним форматима.

Платформа је првенствено намењена будућим средњошколцима и мануелном приступу. На њој се може видети све што се бодује при упису у средњу школу (оцене из последња три разреда, успех на завршном испиту, успех на пријемном испиту, награде на такмичењима за оне године када се то рачунало, бодови добијени на рачун афирмативних мера), као и основни детаљи за сваку од основних и средњих школа на територији државе. Свим подацима појединачно се може приступити преко сајта, али не постоји начин за преузимање свих података заједно, нити начин да се они на било који начин анализирају. Такође, сајт се сваке године ажурира новим подацима, а од старих јавно доступни остају само поједине просечне вредности претходне године, тако да нема начина да се ови подаци упореде са претходним циклусима уписа.

Завод за вредновање квалитета образовања и васпитања издаје годишњи Извештај о реализацији и резултатима завршних испита на крају основног образовања и васпитања (доступни на https://ceo.edu.rs/извештаји-о-реализацији-и-резултатим), који иако је добра почетна тачка, се фокусира на завршни испит и популацију углавном дели по основној школи и, чешће, територијално, уместо нпр. по уписаној школи или подручју рада уписаног смера (оваква методологија је разумљива с обзиром да је циљ Извештаја анализа завршног испита, али иде у прилог чињеници да један извештај годишње, ма колико опширан, никако не може бити довољан).

1.2 Циљ и структура рада

Циљ овог рада је да покаже једну од метода за прикупљање јавно доступних информација, као и на који начин се прикупљени подаци могу употребити. Први део рада се тиче прикупљања података и његово чување у фајловима и у релационој бази података.

Како је сајт са подацима које се односе на упис доживео велики редизајн и модернизацију у току 2017. године, постоје два начина за преузимање података које овај програм подржава у зависности од варијанте сајта са којег се подаци преузимају. Други део рада се односи на обраду података и описује начине за манипулисање са подацима на два начина: учитавањем фајлова са подацима и користећи изграђену базу података. Сви прикупљени подаци се односе на ученике који су уписали средњу школу сем ако није другачије назначено, односно циљ рада није анализирање комплетне популације основаца у датом периоду, пошто је упитно да ли је такав подухват тренутно уопште могућ, а ако јесте онда захтева далеко већи ангажман у односу на то колико користи може да донесе.

Део I:

Прикупљање података

Подаци су драгоцена ствар и трајаће дуже од самих система.

Тим Бернерс-Ли, изумитељ Веба

Глава 2

Појам скреповања

Овде се, као и у остатку рада, због стила и недостатка боље формулације реч ученик употребљава у граматичком мушком роду. Овај израз наравно обухвата и све ученице.

2.1 Шта је скреповање

Скреповање Веба (енгл. web scrapping, у дословном преводу Веб стругање) подразумева било који поступак којим се подаци из формата који је читљив људима пребацују у формат читљив машинама [2]. Веб странице су најчешће писане у HTML-у и CSS-у (од енгл. Hypertext Markup Language и Cascaded Style Sheets), уз опционе процедуралне елементе кроз Javascript, које претраживач чита и приказује кориснику. Садржај се по правилу налази у оквиру HTML или као што ћемо видети у неким случајевима унутар Javascript кода, док CSS носи информацију о изгледу странице. Овај процес је оптималан када су циљна група људи, међутим ови језици су далеко експресивнији и носе значајан вишак информација за потребе анализирања самих података (нпр. начин на који је све визуелно организовано на страници), који треба одстранити.

Најочигледнији метод скреповања је ручни – тако што човек копира, налепљује и организује одговарајуће податке тако да се они касније могу лако учитати користећи неки програм. Ово је, међутим, ретко практично, и користи се само када не постоји други избор, или за мање корекције самих података. Други најједноставнији метод је тражење образаца у тексту/коду и извлачење фрагмената са одговарајућих места (видети, на пример, grep алат на Unix-базираним системима и појам регуларних израза). И овај поступак је незадовољавајући када је реч о страницама које се највећим делом састоје од HTML кода и као такве имају велики број сличних образаца у тексту које је тешко

разликовати. Значајно бољи начин је парсирање HTML-а на начин сличан оном како то претраживачи раде.

2.1.1 HTML и DOM

HTML је језик чији су основни елементи тагови који одређују тип података и могу носити опционалне атрибуте као што су јединствена идентификациона вредност (id) или класа (class), а унутар којих се налази садржај (конкретна синтакса нам у овом случају није значајна). Када се тај код машински „прочита" и интерпретира, добијамо Објектни модел документа (енгл. Document Object Model, или DOM), који у меморији хијерархијски представља све елементе у облику стабла са кореним чвором *document*. Ова структура је много лакша за манипулисање [3], и овде ће се највише користити за долажење до садржаја конкретног елемента на страници. Објектни модел документа је апстрактни интерфејс који није везан ни за један конкретан језик или платформу [4], али ће се у контексту овог рада увек односити на резултат парсирања HTML странице.

2.1.2 Легалност скреповања

Када је реч о прикупљању података без експлицитне дозволе, често се помиње легални аспект и да ли је и када то законом дозвољено. Најчешћи аргумент против овакве технике је угрожавање ауторских права или кршење Услова коришћења сајта, и на то треба посебно обратити пажњу.

У Србији, колико је мени познато, није било већих случајева из ове области, а закон који уређује област је управо Закон о ауторским и сродним правима [5]. Конкретно, у случају овог рада, прикупљани подаци нису оригинална духовна творевина аутора (чл. 2 Закона), него подаци настали у раду органа јавне власти како је дефинисано у чл. 2 Закона о слободном приступу информацијама од јавног значаја [6] у које се по чл. 3 убрајају и све државне основне и средње школе у Србији. Такође, нигде на сајту не постоје експлицитно наведени услови употребе и ограничења која би се односила на скреповање, укључујући и robots.txt фајл који се конвенционално користи за истицање правила понашања свих посетилаца који нису људи, већ скрипте и програми у било каквом облику (видети Стандард за искључење робота [7]).

2.2 Извор података

Као извор података за скреповање користи се http://upis.mpn.gov.rs, који је званичан портал Министарства просвете, науке и технолошког развоја посвећен упису у средње школе чији је оснивач Држава, аутономна покрајина или јединица локалне самоуправе [8]. На порталу се могу пронаћи подаци о сваком уписаном ученику (у анонимизираној форми, тј. под шифром) и свакој основној и средњој школи која је део система, али насупрот тврђењу на почетној страници које обећава „све статистичке податке који могу помоћи у одабиру школе" [8] сматрам да се на њему може видети само делић статистичих података који се коришћењем свих изложених података може извести. Овај рад ће пружити примере за неке од њих и начинити базу доступну свима који имају могућности и вољу да дају свој допринос.

2.3 Структура портала посвећеног упису у средње школе

Портал је доживео редизајн 2017. године, стога је било потребно направити скрепер (енгл. scraper, у значењу „онај који скрепује") у две варијанте, односно ажурирати постојећи тако да може да „разуме" и нову верзију. Основна структура портала је остала иста. Одељак *Подаци о ученицима* омогућава претрагу конкретног ученика по шифри и ово нам за аутоматско преузимање података није од користи пошто може постојати и до милион шифара за мање од седамдесет хиљада ученика колико упише средњу школу сваке године (ову тврдњу ћемо доказати у другом делу, у Глави 5). Одељак *Основне школе* пружа основне информације о свакој основној школи као што су име, место, број ученика и просечни бројеви бодова на крају разреда и на завршном испиту; ови подаци се прикупљају парцијално од 2016. године, а у пуном облику од 2017-те.

Средње школе и образовни профили је најкориснији одељак за скрепер. Он обухвата, за сваки образовни профил, тј. смер средње школе, списак шифара ученика, број бодова и круг уписа за сваког уписаног ученика. Када знамо шифру ученика, можемо је користити да приступимо страници која садржи податке о конкретном ученику, као што су оцене из свих предмета у завршне три године које се бодују при упису и бодови на завршном испиту. И када се порталу приступа ручно, текст којим је исписана шифра је хиперлинк до странице ученика, тако да је без много муке могуће одратити верификацију појединих записа.

Глава 3

Скрепер

Скрепер је, као што је споменуто, програм који прикупља податке са неког сајта. Овај рад ће се састојати од више компоненти, а ово је његов кључни део који омогућава све остале. У овом раду, када се говори о специфичном пакету, класи, интерфејсу, методи или пољу, користи се Javadoc стил, тј. као хијерархијски сепаратор пакета и класа се користи карактер `.`, док се пре методе налази карактер `#`. Сви називи су дословно узети и записани су користећи `monospace фонт`.

3.1 Структура програма

За изворни код који ће се овде коментарисати, видети додатак B.

Комплетан програм је писан у програмском језику Java, верзија 8 и користи Gradle као *build system* који изграђује пројекат, тј. проналази и преузима библиотеке, компајлира код и пакује све у .jar фајл који се може покренути на Java виртуелној машини. Аналогну улогу игра make у екосистему програмског језика C. За разлику од алата Ant и Maven који су нешто старији и чешће се срећу, посебно у старијим програмима и у курикулуму, Gradle је базиран на синтакси програмског језика Groovy уместо XML-а, што чини конфигурацијске фајлове краћим и лакшим за читање. Конфигурацијски фајл се по конвенцији назива `build.gradle` и налази се у кореном директоријуму пројекта. (Детаљан опис Gradle система је ван домена овог рада. За више информација, видети званичну документацију на `https://docs.gradle.org/current/userguide/userguide.html`.)

Код се налази унутар директоријума `src/main/java/` и корени пакет се зове `rs.lukaj.upisstats.scraper`, унутар кога је главна (Main) класа `UpisMpn`. Програм прима један или више аргумената преко командне линије. Аргумент којим се врши преузимање је `dl` (од енглеског *download*) и ово поглавље објашњава како тај процес тече. Друга могућа

команда је `exec`, праћена методама за извршавање, која је детаљније описана у одељку 6.1 и има смисла само када су подаци већ преузети. Сав код који је релевантан за ово поглавље се налази у `download` пакету.

3.2 Принцип рада и дељени код

Већ је споменуто да се портал са кога се преузимају подаци значајно разликовао пре 2017. године у односу на садашњу варијанту. Међутим, принцип којим се до података долази је већим делом исти, што омогућава да је део кода дељен и користи се без обзира која се варијанта преузима. Као заједнички интерфејс се користи `DownloadConfig` и он представља спону између два начина за преузимање. Његове имплементације, `DownloadConfig.Old` и `DownloadConfig.New` пружају, редом, начине да се обави скреповање старог и новог портала.

`DownloadController` контролише процес преузимања, који тече на сличан начин као када бисмо ручно копирали све податке. Прво се преузимају шифре свих смерова и чувају у фајл `smerovi` који се налази унутар директоријума одређеног `File` објектом `DATA_FOLDER`. Ако је преузимање раније започето, односно постоји `save` фајл, из њега се учитава индекс смера чије је преузимање прекинуто и оно започиње испочетка (како и највећи смерови броје мање од 300 ученика, количина посла која се понавља је занемарљива у односу на укупан обим).

Други дељени део кода је `UceniciManager` који одржава редове шифара које треба преузети, преузетих ученика и неуспелих преузимања. У ред у којем се налазе шифре се додају нове у групама (користећи `UceniciManager#add`), где свака група представља један образовни профил, тј. смер, до одређене границе. Када се граница прекорачи, све шифре се шаљу на преузимање (`UceniciManager#download`), а затим се подаци чувају, за сваког ученика у засебан фајл (ово је прилично наивна и неефикасна шема за чување података за обраду, али је у тренутку писања деловала практично за потребе преузимања; касније ћемо видети на који начин се може унапредити).

Чување се одвија у засебној нити користећи `ExecutorService` механизам из Java стандардне библиотеке који поједностављује рад са вишенитним (енгл. multithreaded) програмима, како се не би блокирао процес преузимања. Када се преузимање заврши, нит која је претходно популисала ред са шифрама непреузетих ученика се ослобађа и процес се наставља, све док постоје непреузети смерови (за овај део је заслужан `StudentDownloader#downloadStudentData` који се такође користи независно од године преузимања).

3.3 Стара варијанта – пре 2017. године

Стари сервер је био писан у програмском језику PHP и налазио се на адреси 195.222.98.59 (из неког разлога, сајт је увек правио конекције директно на IP адресу, уместо на домен, тако да и скрепер имитира то понашање). За сваку категорију података, било је неопходно слати нови захтев како би сервер вратио страницу са траженим информацијама. Тако се страница са листом смерова који се налазе у одређеном округу добија једним захтевом, оцене из шестог разреда конкретног ученика другим, оцене из седмог разреда трећим, итд. Конкретни параметри и адресе се могу добити посматрањем захтева које претраживач прави када се порталу ручно приступа, и затим у програму само треба поновити исте са измењеном шифром смера или ученика.

`StudentDownloader` је класа која служи за преузимање шифара ученика. Странице на старом порталу су биле организоване као више нивоа угњеждених табела (што је био популаран начин за дизајнирање сајтова деведесетих година прошлог века [9]), из којих овај програм треба да извуче податке. За парсирање HTML кода који се добија преузимањем користи се Jsoup библиотека која може да изгради DOM у меморији и врати корени чвор, `Document`. Свака ћелија у табели има јединствени секвенцијални `id`, и табела са шифрама има четири колоне, тако да знамо да се шифре налазе у ћелијама обележеним `id`-јевима 1, 5, 9, итд. а бројеви бодова у ћелијама 4, 8, 12, итд.

За потребе селектовања одређеног елемента, Jsoup подржава *CSS селекторе*. Од синтаксних правила битних за овај пројекат, важно је знати да се `id`-јеви (`id` атрибут на HTML елементу) означавају почетном `#`, класе (`class` атрибут) почетном тачком, а размак означава хијерархијско „испод”. Тако нпр. `#id0 .class1 .class2` селектује све елементе класе `class2` који су деца елемената класе `class1`, који су деца елемента чији `id` атрибут има вредност `id0`. Ово је сасвим довољно да дођемо до произвољног елемента ако он има `id` (који је по дефиницији јединствен) или класе елемената кроз коју можемо проћи итерацијом и издвојити оне који нам требају.

Када знамо шифру ученика, конструишемо нову инстанцу класе `Ucenik` чијем конструктору прослеђујемо шифру, и позивамо методу `#loadFromNet`. Круг уписа и укупан број бодова нису видљиви на страници ученика, тако да се они „убацују” користећи `#setDetails` методу (укупан број бодова на страници ученика приказује само збир бодова добијених на основу оцена и пријемног испита, док је на страници смера на којој је листа ученика приказан стваран број, који рачуна и пријемни испит и афирмативне мере). `#loadFromNet` учитава страницу ученика, странице за оцене из сваког разреда и листе жеља за оба круга, ако постоје, и из сваке табеле извлачи адекватне податке. `#toCompactString` и `#loadFromString` дефинишу формат у којем се `Ucenik` чува на диску, тј. начин на који се сва поља серијализују.

`Ucenik` садржи поља у којима се преузети подаци чувају, која су или текстуалног типа (`String`) или мапирају текстуални кључ у текстуалну вредност (`Map<String, String>`). Оваква флексибилност омогућава да поједине вредности недостају, да се на месту где се очекује бројчана вредност појави неки други карактер, или уопште да се било где појави било каква вредност. Иако можда звучи контраинтуитивно, ово је управо жељено понашање. У системима ове скале, чак и ако су савршено пројектовани и реализовани (што углавном није случај), није оправдано претпоставити било коју структуру података која аутору звучи логично, већ је много јефтиније прихватити све улазе при прикупљању, а верификацију претпоставки оставити за процес обраде података и све проблеме решавати „у ходу". Података који недостају ће бити, а вредности ће се јављати чак и на местима где немају семантичког смисла, и нема потребе да то утиче на процес прикупљања који је коректно изведен.

Преузимање комплетних података о основним школама (односно, све сем имена, места и округа) је додато накнадно, у току 2016. године – у првобитној верзији нисам сматрао то довољно битним, нити сам нашао иједан практичан начин да то преузимање изведем. Пошто за сваку генерацију постоји одређени временски прозор у којем су подаци доступни на порталу, није било могуће ретроактивно преузети податке. Класе које врше преузимање и дефинишу структуру података за основне школе се налазе у `.download.misc` пакету. Претпоставља се да су идентификациони бројеви за основне школе који се користе на серверу *скоро секвенцијални*, тј. да је размак између два суседна мањи од 100. Међутим, чак и када се на овај начин преузму подаци, нисам нашао ниједан поуздан начин да се те исте основне школе повежу са ученицима који су их похађали, тако да је употребљивост овако добијених података ограничена.

У фокусу 2015. године када је ово писано су ми били они ученици који су уписали средњу школу у првом кругу, тако да се листе жеља за други круг не преузимају. Такође, подаци о кругу када је ученик уписан нису у потпуности прецизни – не прави се разлика између оних који су уписали смер у другом кругу и оних који су уписани по одлуци окружне уписне комисије. Број уписаних ученика у другом кругу углавном износи око 1500, што чини нешто више од два процента података. (Сви наведени пропусти су исправљени у новијој варијанти, а „пролазност" података на порталу ми не допушта да измене и допуне направим ретроактивно. Управо овај временски оквир и компатибилност као приоритет су главни разлози зашто су неки неоптимални избори у вези са архитектуром и дизајном „преживели" у коду три године и тешко их је у овом тренутку променити.)

У току извршавања програма, само за преузимање података о ученицима, направи се преко 330 хиљада захтева (по шест за сваког ученика), док је за остале типове података овај број компаративно миноран: можемо проценити да укупан број захтева који долази до сервера износи око 350 хиљада. Просечно време извршавања износи око 30 сати, из

чега се добија да се у просеку реализује нешто мање од 3.5 захтева по секунди. У сваком тренутку, отворена је највише једна конекција сервером. Све у свему, иако је процес обиман и дугачак, не може се рећи да на било који начин угрожава или омета рад сервера, пошто чак и серверски оквири којима брзина није први приоритет, попут Apache-а, могу без проблема услужити и за неколико редова величине веће оптерећење [10]. Наравно, требало би избегавати покретање овог програма када се абнормално велика оптерећења сервера очекују, нпр. непосредно након објављивања података, како би се избегла чекања и захтеви који могу остати незадовољени услед преоптерећења и морају бити поновљени (опет наглашавам, овај програм, по дизајну, *не може* значајно да оптерети сервер).

3.4 Нова варијанта – од 2017. године

У неким класама, поједине методе се могу искористити и на новој верзији портала. Како би се избегло дуплицирање кода, користи се *наслеђивање*. Класе које имају суфикс `2017` у имену, сем `OsnovneDownloader2017` и `Osnovna2017`, су поткласе класа истог имена без овог суфикса. Како би ово било могуће, када сам модификовао програм 2017. године, елиминисао сам већину статичких метода које није могуће наследити. Уместо њих, класе које их садрже су добиле заштићени (`protected`) подразумеван конструктор и статичку `getInstance` методу која је намењена за креирање једне инстанце при првом позиву, која се чува у приватном статичком пољу, и враћање сачуване инстанце при сваком следећем позиву (ово је директно инспирисано *singleton* шаблоном).

У овом одељку бих се осврнуо само на то како су подаци организовани на новом порталу. Уместо унутар серије табела, подаци се учитавају користећи Javascript код, унутар кога су дефинисани сви подаци као варијабле. Како бисмо дошли до тих података, нема потребе да тај код извршавамо и чекамо шта ће се појавити на екрану – довољно је да лоцирамо где се код налази и извучемо декларације променљивих, пошто постоји велики број библиотека које разумеју овакве декларације и могу их парсирати. (Технички, такве библиотеке су намењене парсирању JSON формата који је сличан, али није идентичан; међутим, већина библиотека не прави ову дистинкцију.)

Конкретно за пример странице ученика, све променљиве су дефинисане на почетку трећег од позади `script` тага. Ово је много систематичније и поузданије од старе варијанте и јасно је одређено где се која информација налази. Такође, подаци су организовани тако да сваки образовни профил, основна школа, општина, округ, подручје рада и језик има свој интерни идентификациони број. Сви бројеви се преносе при сваком захтеву, што иако је трајање протока, чини посао програма нешто лакшим: пошто су сви подаци на свакој страници, довољно је копирати цео блок са једног места и користити га сваки пут

када треба да од броја добијемо конкретну вредност. Ово нам омогућава да мапирање дефинишемо на једном месту, унутар класе `SmerMappingTools`.

Прва последица боље организованих података која се примећује је значајно мање времена утрошеног за процес преузимања. Пошто се сада за сваког ученика прави један уместо шест захтева, и притом су они појединачно бржи (да ли због боље архитектуре или јачег хардвера, могу само да нагађам), време преузимања је смањено 6-8 пута.

Друга ствар, приметна тек при обради података, је чињеница да су нови подаци потпунији, прецизнији и смисленији. Тако сам, на пример, у старој верзији број бодова са пријемног дефинисао као разлика укупног броја бодова и збира бодова који је добијен на основу оцена, завршног испита и такмичења, што доводи до тога да постоје мале вредности које су резултат (не)заокруживања, или бодови које су заправо резултат афирмативне мере а не пријемног испита, или чак негативне вредности који се јављају у неким случајевима када је ученик уписан одлуком окружне уписне комисије и има заведен укупан број бодова као нула. У новој верзији, сасвим је јасно где се налази број бодова са пријемног испита на страници сваког ученика. На располагању је и више података: подаци о основним школама су систематично преузети и повезани са ученицима, за сваког ученика постоји јасно назначено на основу чега је добио који бод, па чак се и за сваку ставку на листи жеља види колико му се бодова рачуна, пошто ови бодови не морају да буду нужно исти, нпр. ако је полагао више пријемних испита са различитим успехом.

Нова верзија скрепера, уз податке серијализоване у формату који описује `Ucenik2017#toCompactString` метода, чува и оригиналне исечке Javascript кода из кога су оне добијени, за случај да буде неопходно употпунити податке, или верификовати неки податак без приступа оригиналном извору. Податке је могуће учитати фајл по фајл, користећи `Ucenik#loadFromString` (и, аналогно, `Ucenik2017#loadFromString`), али овај процес на класичним хард дисковима може да потраје много више времена него што је то неопходно. Начин како да се ово убрза је спајање свих фајлова у један, што је једна од првих ствари описаних у глави 6, и препоручљиво је извршити је пре било какве обраде.

3.5 Омотачи

Раније је споменуто да се сви преузети подаци чувају у текстуалном формату и образложене су предности овог приступа. Није тешко уочити зашто је текстуални формат непрактичан за обраду, а његово парсирање сваки пут када желимо да дођемо до броја је непотребно скупо. За потребе обраде, дефинисани су *омотачи*, објекти идентичне

садржине али различитих типова података који ће се у остатку рада користити за манипулисање подацима.

Омотач за податке прикупљене користећи стару варијанту скрепера је дефинисан класом `rs.lukaj.upisstats.scraper.obrada.UcenikWrapper`, унутар које се налазе класе `Takmicenje` (чува податке о нивоу такмичења и освојеној награди), `OsnovnaSkola` (садржи податке о похађаној основној школи у којој је ученик завршио осми разред) и `SrednjaSkola` (састоји се од података о уписаном смеру). Сем конвертовања текстуалних података у нумеричке вредности и обрађивања случајева када неки податак не постоји, омотач рачуна и чини лако доступним просеке и тачне бројеве бодова за сваку категорију која се бодује (неки од ових података су додати и на сајт 2016. године).

Омотач за податке прикупљене 2017. године је дефинисан класом `rs.lukaj.upisstats.scraper.obrada2017.UcenikW`, и за разлику од „старијег брата", не садржи потклассе које дефинишу податке о похађаној школи и уписаном смеру, већ су оне издвојене у `.obrada2017.OsnovnaW` и `.obrada2017.SmerW`. Ове класе такође у суштини „чисте" податке и рачунају корисне додатне вредности.

Глава 4

Сервер и база

4.1 Улога сервера

Основни циљ скрепера је управо скреповање, тј. преузимање података, док је њихова обрада на другом месту. Иако је могуће написати релативно напредне методе за обраду и покренути их користећи интерфејс који главни програм подржава, он и даље има неколико ограничења. Први проблем је учитавање: подаци могу да се учитају или из појединачних фајлова, што је изузетно споро у великим количинама, или из једног спојеног фајла, што може сместити више података у меморију него што је неопходно. Друго, програм је конципиран да ради на једној генерацији у било ком тренутку; иако је могуће учитати податке за више генерација истовремено и оперисати над њима, овај процес би могао бити много лакши. Напослетку, пошто је оригинална замисао да се програм не покреће више од једанпут годишње, перформансе често нису биле приоритет.

Сви ови проблеми се могу решити увођењем релационе базе података.[1] Прва улога сервера ће бити управо изградња ове базе користећи модел података који скрепер пружа, а затим ћемо показати како се овај сервер може употребити за реализовање сервиса који користе ову базу. Разлог за коришћење комплетног сервера уместо, рецимо, писања метода који би на неки од стандардних начина (нпр. кроз JDBC) оперисали над базом је чињеница да велики број ствари долази уграђено, тако да се избегава директан приступ бази тамо где то није неопходно.

Сервер је писан у Play! Framework-у, који је оригинално део Scala (програмски језик који се компајлира за JVM, делом инспирисан Javom) екосистема, али подржава и Java

[1] Могуће је, наравно, било да се подаци у старту памте у релационој бази, али 2015. када сам започео овај пројекат нисам га тако конципирао. Касније је свакако било неопходно да се напише код за портовање, па није било разлога да се оригинални пројекат модификује.

језик. Уместо Gradle-а користи SBT (видети https://www.scala-sbt.org/1.x/docs/Getting-Started.html), има уграђен ORM (више о томе у одељку 4.3) и има нешто другачију структуру директоријума. С друге стране, и даље је у питању Java код, тако да нису неопходни огромни уводи за његово разумевање. Debug мод се покреће командом `./sbt run`, а цео пројекат се компајлира и пакује са `./sbt dist`, који креира .zip фајл у директоријуму `target/universal/`.

4.2 База података

За базу података која ће се овде коментарисати, видети додатак Б.

У бази се чува шест типова ентитета: ученик, смер, основна школа, такмичење, листа жеља и пријемни испит. Са изузетком пријемног испита који су издвојени тек у 2017. години, сви имају три подтипа, за сваку годину прикупљања. Између 2015. и 2016. године постоје мање разлике, док су оне израженије у новијој варијанти, стога има смисла раздвојити их у различите табеле.

Ентитет ученик садржи оцене за сваки предмет од шестог до осмог разреда (у формату `{име_предмета}{бројразреда}`), просеке (у формату `{име_предмета}_р`), бројеве бодова као и укупан број бодова, и страни кључ који указује на завршену основну школу и уписани смер. Истоветни подаци о оценама и бодовима се налазе и унутар основних школа и смерова, и они представљају просечне вредности истоветно званих атрибута ученика који су ту школу похађали/тај смер уписали. Уз то, доступни су и основни подаци о свакој основној школи и смеру, као што су место, округ, број ученика, а за смерове и средња школа, подручје рада, квота и трајање. Префикс `svi_` у називима атрибута који се односе на податке из 2017. године означава да се тај податак односи на све ученике, а не само на оне који су уписали средњу школу. Такмичења, пријемни и листа жеља садрже број бодова, страни кључ који упућује на ученика и детаље попут редног броја жеље или нивоа и награде на такмичењу.

Приметићемо да су неки од овде наведених података изведени, тј. да се могу добити директном обрадом осталих података. Иако се то у општем случају сматра лошом праксом, ова база има неколико специфичних особина. Прво, неке ствари би било непрактично рачунати путем SQL-а, нпр. формула за просек оцена није тривијална када се узме у обзир да се број оцена по разреду варира од 7 до 15 међу ученицима, посебно када знамо да неке од ових информација већ постоје на самом сајту. Битније, ова база није променљива.[2] Подаци у њој су финални и нема смисла мењати их, већ је једина валидна операција додавање, тако да нема потребе да бринемо о њиховом ажурирању.

[2] У току развоја, ако се примети грешка, може бити потребе да се одређени подаци промене. И у овом случају, након што се грешка исправи, лакше је обрисати све податке за одређену генерацију и затим их поново учитати него писати посебан код који би преправљао податке редом.

4.3 Структура сервера

За изворни код који ће се овде коментарисати, видети додатак B.

Како не би било непотребног дуплицираног кода, сервер укључује целокупан скрепер као библиотеку. Овај процес је у највећој мери аутоматизован: комплетан код се налази на `https://github.com/luq-0/UpisScrapper`, што омогућава коришћење `https://jitpack.io` сервиса, који пакује и објављује пројекат као библиотеку. Све што је преостало је њено укључивање у пројекат сервера, у фајлу `build.sbt` који се налази у кореном директоријуму (последња линија текста). Надаље можемо користити све јавне методе и поља из скрепера у коду сервера.

Унутар `conf/` директоријума се налазе конфигурацијски фајлови. Најважнији део `conf/application.conf` фајла су параметри за конектовање на базу – када податке учитате у сопствену базу, потребно је подесити корисничко име, лозинку и адресу базе у одговарајућим `db.default.*` пољима. `routes` фајл дефинише руте, тј. одређује која ће се метода позвати за сваку путању. Нпр. ако корисник посети `{адреса_сервера}/query?initial=asdf`, позваће се `controllers.Index#query` са аргументом `asdf` (за случај да `initial` параметар није прослеђен, методи се шаље празан `String`).

Унутар `conf/evolutions.default` директоријума се налазе еволуције за `default` базу. Овај механизам нам омогућава да ажурирамо структуру базе (нпр. да додамо нове табеле сваке године), као и да се крећемо унапред и уназад кроз различите верзије базе. Свака еволуција се састоји из два дела: `Ups` који дефинише нове измене и `Downs` у којем се налази код за враћање из измењеног у старо стање. Оба дела се састоје из SQL наредби које модификују структуру базе, попут `CREATE TABLE`, `DROP TABLE` и `ALTER TABLE`. Фајлови се извршавају секвенцијално. Први фајл, `1.sql`, је аутоматски генерисан.

Програмски код се налази у `app/` директоријуму. Пакет `controllers` дефинише класе које примају захтеве (од рутера), `models` одређују структуру података, а `views/` презентују странице кориснику.

4.4 Изградња базе

4.4.1 ORM и модел

Једини SQL код написан за потребе овог пројекта је онај за еволуције, и то не рачунајући прву. Остатак генерише *објектно-релациони мапер* (ORM, енгл. object-relational mapper) који класе из објектно-оријентисаног језика као што је Java преводи у SQL табеле и,

аналогно, објекте у инстанце ентитета, односно редове у табели. Објектно-оријентисано програмирање није базирано ни на каквом математичком формализму (иако постоје покушаји да се развије рачун који би га описао [11]), док се релационе базе података темеље на релационој алгебри. Ова два система нису потпуно еквивалентна, стога ни мапирање не може бити савршено [12]. Међутим, ако се на уму има жељени резултат, могуће је саставити класе које би га генерисале без већих проблема. Подразумевани мапер за Java пројекте писане у Play! Framework-у је Ebean (http://ebean-orm.github.io/), који се користи и у овом пројекту.

Класе из којих се генеришу табеле налазе се у пакету `models`. Све класе које немају годину као суфикс су суперкласе и садрже поља која су заједничка за све ентитете тог типа, без обзира на генерацију и оне носе анотацију `@MappedSuperclass` што означава да не Ученик није завршио основну школу, те не може конкурисати за упис у средњу школу постоји табела у физичком моделу за њих. Оне проширују класу `com.avaje.ebean.Model` која је „основа" за све ентитете. Класе које их наслеђују наслеђују њихова поља и имају опцију да додају своја, у случају да су се за ту генерацију прикупљали још неки додатни подаци (ово је посебно изражено код података преузиманих са новог портала, тј. за генерацију која је завршила основну школу 2017. године). Оне су означене анотацијама `@Entity` и `@Table` са атрибутом `name` који означава име табеле у којој се подаци смештају.

Унутар класа, сваки примитивни податак и `String` добија своју колону примитивног SQL типа. Ако је поље класа која је уједно и ентитет (носи `@Entity` анотацију), креира се колона која ће служити као страни кључ ка табели која је дефинисана датим ентитетом и треба је обележити `ManyToOne` или `OneToOne` анотацијом. Ако је поље листа ентитета, креира се *one-to-many* веза са ентитетом који ће садржати страни кључ ка овом ентитету. [3] *Many* крај може експлицитно дефинисати референцу ка *one* крају тако што ће садржати поље адекватног типа. *One-to-many* и, са друге стране, *many-to-one* везе морају бити обележене адекватним анотацијама над пољима.

Узмимо класу `Ucenik2017` као пример. Унутар ње, постоје `@ManyToOne` поља за основну школу и уписани смер. (Унутар ова два ентитета не постоји одговарајућа листа, јер нема потребе учитавати и све ученике из базе када се учита школа – углавном је довољно да се само један крај везе спецификује, а ORM ће решити други.) Али, пошто ученици углавном имају више жеља, потребна је листа жеља (типа `List<Zelja2017>`, која је обележена са `@OneToMany`. Овде експлицитно наводимо и други крај, тако да је потребно код листе жеља експлицитно назначити да је други крај везе заправо поље `Zelja2017#ucenik` користећи `mappedBy` атрибут.

[3] Технички, креира се *many* крај на повезаном ентитету, који такође може садржати листу и тако дефинисати *many-to-many* везу. Аутоматско разрешавање овога доводи до креирање нове табеле и често није оптимално, па се у овом пројекту не користи.

ОРМ аутоматски генерише и извршава наредбе за чување, ажурирање и брисање података из базе када се позове `Model#save`, `Model#update` или `Model#delete`. `Model.Finder` служи за проналажење одређене инстанце, и у сваком ентитету постоји одговарајуће статичко поље `finder` помоћу кога се може приступити садржају базе (еквивалентно `SELECT` упитима).

4.4.2 Учитавање података, ламбде и рефлекција

Методе за учитавање података се налазе унутар `controllers.Init` класе. И овде постоје две варијанте – за податке пре 2017. године, метода `#populateDb`, а за оне касније `#populateDb2017`. Како би се спречиле било какве друге акције током учитавања података, када је `Init#INIT_PHASE` постављен на `true`, дозвољено је само извршавање метода из `Init` класе. У супротном, када је `Init#INIT_PHASE` `false`, није дозвољено извршавати метода за иницијализацију.

Учитавање података се обавља у неколико трансакција на бази. Позив `com.avaje.ebean.Ebean#execute` означава једну трансакцију. Ова метода као аргумент прима *ламбду*. Ламбда се може посматрати као анонимна метода. Унутар заграда се налазе параметри, ако постоје, затим следи „стрелица” (`->`), па једна наредба или тело методе. Ако је ламбда облика `(arg) -> Class.method(arg)`, она се може заменити референцом на методу облика `Class::method`. Овакве методе, које као аргументе могу примати ламбде, су еквивалент функцијама вишег реда из функционалног програмирања.

Узмимо као пример линију кода из `Init#populateDb2017` методе: `Ebean.execute(() -> svi.forEach(Ucenik2017::create));`, где је `svi` променљива типа `Set<UcenikW>`. Она извршава трансакцију у којој се над променљивом `svi` позива метода `forEach` која над сваким елементом извршава одређену операцију. `forEach` такође као аргумент прима ламбду, и њој се прослеђује референца на методу `Ucenik2017#create`. Ово резултује у позивању методе `Ucenik2017#create(UcenikW)` за сваки члан скупа `svi`, унутар једне трансакције на бази. У случају да дође до изузетка или грешке при извршавању, аутоматски се извршава `rollback` и база се враћа у првобитно стање. Сличан поступак се извршава и за сваки смер и основну школу, независно од генерације, и сви они имају одговарајућу статичку методу `create`.

Приметићемо да унутар сваког ентитета постоји доста сличних поља. Рецимо, за сваки предмет, називи поља су истог облика, и оцене из математике се чувају унутар поља `matematika6`, `matematika7` и `matematika8`, док је њихов просек смештен у `matematika_p`. Ручно постављање ових поља би била прилично заморна и репетитивна радња, посебно када постоји јасан процедурални поступак за њихову иницијализацију. У случају генерације чији су подаци прикупљани 2017. године, метода која пребацује податке из облика

у којем их библиотека чува у облик који се серијализује у базу се зове `models.Ucenik#populateGrades2017`.

Скрепер чува податке о оценама у мапи која има текстуални кључ и целобројну вредност. Осим у пар изузетака, поље у ком вредност треба да се чува ће имати назив `{кључ}{разред}`, где је разред `6`, `7` или `8`. Механизам којим је могуће у току извршавања приступити неком пољу, методи или класи чије се име не мора знати унапред се назива *рефлекција*. Почетна тачка је оно што знамо, а то је класа: знамо да се поља налазе у класи чија инстанца је прослеђена методи `Ucenik#populateGrades2017`, и зовемо `#getClass` над тим објектом како бисмо добили референцу на класу (објекат класе `Class`). Затим, над `Class` објектом извршавамо `#getField(String)`, где је једини аргумент име поља, како бисмо добили референцу на поље. Коначно над пољем извршавамо `#setInt(Object, int)` коме прослеђујемо `int` жељену вредност, и инстанцу чијем пољу ту вредност желимо да доделимо.

Овај механизам се користи и код рачунања просечних вредности и у ентитетима који се односе на ученике и у онима који се односе на школе и смерове. За последња два су задужене методе `Init#populateSchoolAverages` и `Init#populateAveragesInner`. Једина разлика је што не постоји мапа из које се извлаче вредности, већ се користи `Class#getFields` како би се добио низ поља из којих се састоји основна школа/смер. За свако поље које је типа `double` (проверава се позивом `Field#getType` и упоређивањем враћене вредности са `double.class`) и постоји поље истог имена типа `int` или `double` у класи ученик, рачуна се просек вредности поља сваког ученика који је похађао ту школу/уписао тај смер и уписује се у одговарајуће поље школе или смера.

4.5 Пример употребе – упити и базично плотовање

Пример апликације која се може направити када постоји функционална база је приложен у Додатку B, као део изворног кода сервера. У питању је веб платформа која допушта кориснику на поставља упите серверу и на њих одговори или једним бројем или графиконом који на две или три осе представља дате податке. Већи кружићи на плоту означавају већи број ентитета са одређеним вредностима особина које су дате осама.

Језик којим се постављају упити је инспирисан SQL-ом, користи српске речи, а преводи се у SQL који се на бази извршава. Превођење се извршава у класи `controllers.Parser`. Један упит се може превести у више SQL упита, а ако се они плотују постоји опција да њихови резултати буду различито обојени. Када се сервер покрене, подразумевано понашање је преусмеравање на путању `/query` која служи као почетна тачка за ову апликацију.

Ова функционалност је дата као пример употребе сервера, и њен детаљан опис није у домену овог рада. Изворни код садржи README фајл са више информација, а тренутна верзија се може наћи и на https://github.com/luq-0/UpisStats.

Овакав приступ је једноставнији за крајњег корисника и може да пружи лепу визуелизацију података, али није довољно робустан за било какву иоле озбиљнију анализу података. Начини на које се може приступити свим подацима без ограничења, и кроз скрепер и кроз базу, су изложени у следећем делу.

Део II:

Обрада података

Капитална је грешка теоретисати без података.

Шерлок Холмс, у роману „Црвена нит”
(сер Артур Конан Дојл)

Глава 5

Непосредна обрада података

5.1 Кроз скрепер (Java)

Сетимо се да су омотачи који су корисни у случају обраде података дефинисани у оквиру скрепера. Ово значи да ако желимо да их искористимо, код за обраду мора на неки начин укључивати пројекат који садржи скрепер. За сада, код који обрађује податке ће се налазити у истом пројекту, али у засебним паковањима `rs.lukaj.upisstats.scraper.obrada` и `rs.lukaj.upisstats.scraper.obrada2017`.

5.1.1 Извршавање метода за обраду

Архитектура пројекта дозвољава да било ко ко има приступ коду дефинише своје методе за обраду и изврши их прослеђивањем параметара `exec {назив_методе}` при покретању програма, где је {назив_методе} назив методе коју треба извршити. Могуће је проследити више назива метода раздвојених размаком, у ком случају ће оне бити извршене секвенцијално. Методе које се извршавају на овај начин морају бити статичке и не треба да примају аргументе (могуће је, наравно, да оне читају са стандардног улаза).

Само извршавање метода се врши механизмом који смо већ увели у овом раду у одељку 4.4.2, рефлекцијом. Међутим, потребан је један додатни корак: све класе у којима се налазе методе које је могуће извршити на овај начин треба додати у скуп `obrada.Exec#executableClasses`, ручном модификацијом методе `obrada.Exec#registerExecutables`. Постоје два разлога зашто је овај наизглед сувишан ручни процес неопходан. Прво, не постоји начин да се рефлекцијом добије референца на неки пакет, који у Java свету не представља много више од обичног директоријума. Друго, да бисмо могли да приступимо некој класи рефлективно, она мора бити учитана, а класе се у Java виртуелну

машину учитавају тек у оном тренутку кад су неопходне. Пошто се до тренутка рефлективног извршавања методе ниједан део кода из класе не извршава, она за виртуелну машину ефективно не постоји и није могуће приступити њеним члановима.

Када се `main` метода позива са првим параметром `exec`, она остале параметре прослеђује `Exec#doExec(String...)` методи која пролази кроз све регистроване класе и тражи статичку методу без параметара са тим именом, коју затим покреће.

5.1.2 Учитавање података

Прва ствар коју треба урадити са свеже преузетим подацима је спојити нешто више од 60 хиљада фајлова са подацима о ученицима у један, како би се убрзало свако будуће учитавање. Метода која ово ради се налази у `Exec#merge` и позива се командом `exec merge`. Постојеће методе за учитавање података претпостављају да су подаци спојени на овај начин.

Како су омотачи за податке другачији, тако су и методе за њихово учитавање различити. За генерације 2015. и 2016, постоји `obrada.UceniciGroupBuilder` који омогућава нешто грануларнију контролу ученика који ће бити учитани, али очекивање је да ће се за учитавање података углавном користити `obrada.UceniciGroup#svi` статичка метода. Излаз ове методе је `UceniciGroup`, што је суштински `HashSet<UcenikWrapper>`, проширен са неколико корисних метода. За податке из 2017. године, користи се `obrada2017.UceniciBase#svi`.

Ова два начина представљају релативно „висок” поглед на учитавање података. Могуће је, наравно, приступити и директно подацима, кроз `obrada.FileMerger#loadFromOne` за ученике и основне школе (у новој верзији) и `download.Smerovi#loadFromFile`, што ове методе интерно и користе. Главни изазови при учитавању података су како смањити количину и број читања са диска (пошто оба утичу на брзину) и што ефикасније парсирати текст у корисне податке. Прва ствар је највећим делом решена спајањем свега у неколико фајлова и кодирањем често коришћених фрагмената текста (видети, рецимо `download.UcenikUtils.PredmetiDefault`). Што се друге тиче, највећи проблем је у подразумеваној методи за парсирање текста у Javi, `String#split`, која је много моћнија, и самим тим спорија него што је у овом случају потребно (разлика се, наравно, не примети ако се обрађују мање количине података са неколико десетина хиљада сплитова; међутим, постаје значајна како овај број превазилази милионе). [1] Стога је било неопходно развити

[1] Овде вреди споменути и `java.util.StringTokenizer` који иако званично застарео (https://bugs.java.com/bugdatabase/view_bug.do?bug_id=4418160), може бити користан у случајевима да је могуће избећи његове багове.

мали токенизер у класи `utils.StringTokenizer` који се екстензивно употребљава унутар омотача и у овом контексту га треба користити пре него библиотечне методе.

5.1.3 Основни примери обраде

Када су коначно сви подаци и архитектура на месту, време је да се из њих извуку неке корисне информације. Сама анализа података је периферни део овог рада. За извођење било каквих озбиљнијих закључака, потребно је да рад садржи осмишљену методологију и да изврши саму анализу, што би, ако би се додало на већ постојећи материјал, чинило овај рад преобимним. Уместо тога, основни циљ је да се постави темељ, а делови који следе су примери употребе тог темеља на базичним задацима.

Класа `exec.Osnove` садржи методе које приказују како доћи до основних особина података. `Osnove#brojUcenika` на стандардни излаз исписује број ученика за који постоје подаци у свакој генерацији. Тако можемо видети да за 2015. имамо податке за 66338 ученика, од укупно 68419 колико их је приступило завршном испиту у јунском року [13], за 2016. 65274 од 68177 [14] и за 2017. 66130 од 67673 [15]. Није могуће са сигурношћу тврдити из ког разлога фале подаци за мањи број ученика, али претпоставка је да они нису уписали средњу школу у јунском року, стога се њихови подаци нису налазили на сајту за упис у средње школе.

У истој класи се налази помоћна приватна метода `Osnove#prosekSvi(ToDoubleFunction, ToDoubleFunction)` која као аргументе прима два мапера – први дефинише како се ученик из старог модела (2015. и 2016. година) претвара у double вредност, а други ради идентичну операцију за нови модел. Када се мапер примени, упросечавају се добијене double вредности и резултат се штампа на стандардни излаз. Ово је суштински функционалан приступ програмирању и практичан је када се ради са много података, а велики део стандардних операција је подржан у стандардној библиотеци кроз `java.util.Stream` класу. Командама `exec prosekOcene` и `exec prosekZavrsni` на излаз се штампају просеци оцена и завршних испита по годинама (ове методе су такође дефинисане у класи `Osnove` и интерно позивају `Osnove#prosekSvi` методу).

5.1.4 Табеле

Стандардни излаз је само један од начина да се подаци прикажу кориснику. Он је добро решење када постоји мали број података, али постоје случајеви када желимо да добијемо нешто већи и лепо форматиран излаз. У случају да је могуће тај излаз представити табуларно, идеално би било да излаз наше методе за обраду буде Excel (.xlsx) табела. Java не подржава креирање ових табела ниједном од стандардних метода, али постоји

више библиотека који тај посао могу да обаве. У овом пројекту се за испис табела користи Apache POI, који сем табела подржава и друге Office formate.

Метода `obrada.Spreadsheets#writeXSSF(File, String[][])` уписује податке из String матрице у прослеђени фајл као .xlsx табелу. У класи `obrada.Teritorijalno` налазе се методе које рачунају просеке по окрузима и местима и уписују резултате у табелу. Тако можемо видети да су 2016. највишу просечну оцену (у просеку) имали ученици који су завршили основну школу у нишавском округу, чак 4.328, док не постоји округ чији је просек општег успеха испод 3.94. Када се погледају просечни бројеви бодова на пријемном, може се уочити да су прва два места иста као и код оцена (нишавски 19.18 и пчињски 19.06), али је занимљиво да је у призренском округу, где је просечна оцена трећа највећа, скоро 4.24, просек бодова на завршном испиту убедљиво најгори: свега 9.285 од максималних 30. Ово нужно повлачи питање да ли су оцене адекватно мерило знања на републичком нивоу, и колико (и зашто) је оправдано да оне носе највећи број бодова, за чији одговор је неопходна засебна анализа.

5.2 Над базом (SQL)

Иако концептуално базичнија, обрада података кроз скрепер је често спора, и за извршавање и за куцање. Пошто смо у одељку 4.4 поставили базу података, већину ових ствари можемо урадити у пар линија SQL-а и добити идентичан резултат много брже.

Бројеви ученика се могу тривијално добити упитом

```
select 2017 as generacija, count(*) from ucenici2017
union select 2016, count(*) from ucenici2016
union select 2015, count(*) from ucenici2015;
```

и тако се уверити да се и у бази заиста налази исти број података као и у фајловима. Слично за просеке, једина потребна измена је `count(*)` одговарајућом функцијом (нпр. `avg(prosek_ukupno)`). Занимљиво је видети како просек варира између уписаних смерова у односу на подручје рада. Ово се може урадити упитом

```
select podrucje, avg(prosek_ukupno) as prosek,
stddev(prosek_ukupno), sum(broj_ucenika) as "učenici"
from smerovi2017
where broj_ucenika>15
group by podrucje
order by prosek desc;
```

који даје следећи излаз

podrucje	prosek	stddev	učenici
zdravstvoisocijalnazastita	4.73231565859952	0.297691336882488	5633
gimnazija	4.68680371427543	0.253204961077332	16181
kulturaumetnostijavnoinformisanje	4.24381746031746	0.238293789870836	420
ekonomijapravoiadministracija	4.20735642474531	0.348277753423078	7785
elektrotehnika	4.13050006726207	0.545924967736679	6974
hidrometeorologija	4.06727777777778	0.261865211299413	60
geodezijaigradjevinarstvo	3.83675831018518	0.384685731216569	1592
saobracaj	3.82960525626415	0.535880496961179	3603
hemijanemetaliigraficarstvo	3.82381498697439	0.30181393595203	2175
trgovinaugostiteljstvoiturizam	3.64542551589161	0.492473710339965	4470
poljoprivredaproizvodnjaipreradahrane	3.55875164281608	0.34933710194616	2869
masinstvoiobradametala	3.53225656713681	0.391168046713721	4887
sumarstvoiobradadrveta	3.43830385667377	0.40098673865853	579
geologijarudarstvoimetalurgija	3.37249831649832	0.350957008831048	327
ostaladelatnostlicnihusluga	3.36554173630954	0.304710464546345	679
tekstilstvoikozarstvo	3.28472091825989	0.306707049913011	710

Намеће се питање, обзиром на незанемарљиву разлику у старту, колико су смерови уопште уједначени и да ли се њихове оцене касније могу директно поредити (као што се ради при упису на факултете)? Важније, из ког разлога уопште настаје оволика разлика на тако широкој категоризацији као што је подручје рада? Сличан упит се може извршити за оцене за сваки појединачни предмет, резултате завршног, или неко географско категорисање слично оном у примеру из одељка 5.1.4; могућности су практично неограничене.

При писању упита постоји пар специфичности на које треба обратити пажњу. У претходном, у `where` клаузули стоји услов који занемарује смерове са мање од 15 уписаних ученика; ово је вештачка граница чија је једина сврха да избаци смерове који ће (највероватније) бити расформирани услед недовољног броја ученика. Овакви смерови често превише утичу коначни резултат и генерално су гори од оних који упишу више ученика (ово се такође може проверити упитом). Треба пазити и на број предмета, који није свуда исти и варира од три (ШООО Обреновац, 8. разред, 2016. година) до петнаест (појединi ученици припадници националних мањина који уче српски као други матерњи језик). Нова варијанта скрепера ово решава на најоптималнији могући начин и правилно узима у обзир све вредности са сајта, али се у старијим подацима могу пронаћи ученици чији је просек мањи од два, јер су на сајту уместо празних оцена стојале нуле. Такође, мора

се узети у обзир да оцена из владања може бити и јединица: 2017. године 165 ученика имало је закључену јединицу из владања у барем једном од разреда.

Сем ових објашњивих, постоје и неке неочекиване аномалије: 2016. за ученика са шифром 223492 нису уопште унете оцене (уписан по одлуци ОУК), док у 2017. постоји шест ученика који немају уписане податке о полаганом завршном испиту (182032, 205680, 114339, 120101, 316117, 188860). Упркос обавештењу на сајту које тврди да *Ученик није завршио основну школу, те не може конкурисати за упис у средњу школу*, сви су уписани по одлуци ОУК. Можда још чудније, број бодова за ученика 163700 је испод теоретског минимума, јер нису унете никакве оцене за осми разред (он је уписан у другом кругу, по првој и јединој жељи), а за још шест фале оцене из математике у осмом разреду (169542, 965611, 199108, 201805, 869154, 247545). Иако је у теорији могуће да овакве грешке настану услед багова у скреперу, у пракси се показало да то није случај, већ су такви подаци преузети са сајта (сви овде наведени примери су и ручно проверени). Ове и сличне ствари само указују на то да систем није савршен и да се дешава да поједини примери делују бесмислено. Стога, најбоље је да се при анализи усвоји дефанзиван приступ и покушају да се формулишу и примене сви неопходни услови, ма колико они здраворазумски звучали.

Глава 6

Процедурална обрада података

Понекад са подацима желимо да урадимо нешто што није могуће или није практично у SQL-у и потребан нам је неки класичнији програмски језик. Овде ћемо прво видети један овакав пример, симулацију уписа, као део пројекта скрепера, а затим ће бити изложено како приступити бази у програмском језику Python и учитати податке који се касније могу обрадити. Python је чест избор за обраду података због његове концизности и квалитетних библиотека чији је циљ управо статистичка обрада и визуелизације.

6.1 Симулације

Скрепер може симулирати упис у средње школе на основу успеха у основној и података о квотама за сваки смер, и код за то се налази у класи `obrada2017.Simulator`. Конструктор симулатора прима два аргумента: први, типа `Simulator.RankingMethod` одређује како се ученици рангирају, а други, `Predicate<UcenikW>` одређује за које све ученике се врши упис. Имплементација `RankingMethod`-а може дефинисати арбитраран начин за расподелу бодова и приоритета, и тако одговорити на питања као што је *Како би упис изгледао када би систем бодовања био другачији?*, док се предикатом можемо ограничити да уписујемо само ученике са одређеном особином (тј. други аргумент конструктора се користи као филтер).

Симулација се врши у методи `Simulator#simulate`. Алгоритам је прилично наиван, али за потребе повремене симулације обавља задовољавајућ посао. Користимо помоћну класу `UcenikZelja` која се састоји од једне инстанце `UcenikW`-а и једне његове жеље (`UcenikW.Zelja`). За сваку жељу сваког ученика креирамо по једну инстанцу ове класе и стављамо их у једну листу коју сортирамо помоћу `RankingMethod`-а. Тим редом покушавамо да упишемо ученике у жељене смерове. Међутим, проблем настаје ако се за

неког ученика жеља која је на његовој листи жеља касније у заједничкој листи нађе испред неке раније – у том случају, очекујемо да ученик буде уписан по својој листи жеља, док алгоритам бира заједничку. Ако до таквој случаја дође, када дођемо до погрешно уписане жеље, исписујемо га из погрешног смера и уписујемо у праву, а затим крећемо итерацију кроз заједничку листу испочетка. Асимптотска анализа најгорег случаја није тривијална, међутим пошто су овде у питању реални подаци а не патолошки случајеви просечно време извршавања ће бити значајно краће од најгорег, с обзиром да се не очекује да често долази до корекције када је неопходно ресетовати итератор.

Метода `Simulator#verifySimulation` је тест који служи да упореди дату симулацију са стварним подацима и испише број разлика. У класи `obrada2017.DefaultSimulation` је дат `RankingMethod` који се користи при упису у средње школе и `DefaultSimulation#defaultSim` метода која симулира први круг уписа са потпуном тачношћу (у односу на званичне податке), што је доказ да је симулатор исправно написан и да су коришћени подаци комплетни.

6.2 Интерфејсовање са базом – пример у Python-у

Једна од великих предности података који се чувају у релационој бази је универзалност приступа – сваки озбиљнији програмски језик има библиотеке које знају како да читају податке из њих. У Python-у, користио сам `sqlalchemy` са `psycopg2` за конектовање на базу и `pandas` за учитавање података из базе у `DataFrame` којим се може манипулисати.

Ако претпоставимо да се база налази на localhost-у, на уобичајеном порту (5432), да се зове `upisdb`, и приступа јој се са корисничким именом `user` и лозинком `pass`, код који се повезује на базу изгледа овако:

```
import sqlalchemy as s
eng = s.create_engine("postgresql+psycopg2://user:pass@localhost:5432/upisdb")
```

док би помоћна метода која може да извршава основне врсте упита (селекцију и пројекцију) била:

```
import pandas as pd
def query(cols, table, where="1=1", order="id"):
    return pd.read_sql_query("select " + cols + " from " + table + " where "
            + where + " order by " + order, eng)
```

Наравно, као аргумент `read_sql_query` функцији се може проследити арбитраран SQL код. Повратна вредност ће бити `DataFrame`, тип података из `pandas` библиотеке који има бројне помоћне функције, али ако је намера вршити било какве сложеније математичке операције, препоручљиво је претворити податке у `numpy` низ методом `numpy.array(df)` где је df `DataFrame` због брзине. За крај, покажимо како изгледа код који исцртава хистограм општег успеха за све три генерације:

```
import matplotlib.pyplot as plt
def plot_histogram():
    uc2015 = query("prosek_ukupno", "ucenici2015", "prosek_ukupno>=2")
            ['prosek_ukupno'] #treba nam samo ova kolona
            #pandas po difoltu vraća i indeks kao dodatnu kolonu
    uc2016 = query("prosek_ukupno", "ucenici2016", "prosek_ukupno>=2")
            ['prosek_ukupno']
    uc2017 = query("prosek_ukupno", "ucenici2017", "prosek_ukupno>=2")
            ['prosek_ukupno']

    plt.hist(uc2015, bins=36, range=(2, 5), histtype='step', density=True,
            stacked=True, color='blue', linewidth=1.2)
    plt.hist(uc2016, bins=36, range=(2, 5), histtype='step', density=True,
            stacked=True, color='green', linewidth=1.2)
    plt.hist(uc2017, bins=36, range=(2, 5), histtype='step', density=True,
            stacked=True, color='red', linewidth=1.2)
    plt.show()
```

На овом плоту је интересантно приметити „скок” код просека 3.5 и 4.5, што су границе за виши успех. Али, оно што је још уочљивије је непропорционално велик број ученика чији је општи успех 5.0. Једноставним SQL упитом се добија и њихов конкретан проценат:

```
select 2017 as generacija,
    1.0*(select count(*) from ucenici2017 where prosek_ukupno=5) /
        (select count(*) from ucenici2017) as "procenat 5.0"
union select 2016,
    1.0*(select count(*) from ucenici2016 where prosek_ukupno=5) /
        (select count(*) from ucenici2016)
union select 2015,
    1.0*(select count(*) from ucenici2015 where prosek_ukupno=5) /
        (select count(*) from ucenici2015)
```

generacija	prosek 5.0
2017	0.15722062603961893241
2016	0.16020161166773906915
2015	0.15755675480116976695

На сличан начин се може доћи и до осталих корисних информација. Визуелизације су често моћан алат који може указати на специфичне детаље које је потребно истражити, а затим је неопходно и математички показати да је интуиција иза њих тачна.

Глава 7

Закључак

Сваки од показаних примера може бити одељак за себе. Овај рад не пружа одговоре на суштинска питања као што су *зашто* су уочени такви подаци или *како* су они настали, већ само констатује да одређени феномени постоје. Међутим, значај овог пројекта је што омогућава да се на та питања да одговор и ван званичних извештаја, и надам се да ће у будућности бити користан било коме ко пожели да се бави анализом на овај начин прикупљених података.

Такође, ово је леп пример да није практично ограничити се на само један језик или платформу за било који нетривијалан пројекат. Изложена су три програма која се надовезују један на други и чине једну велику целину, и кроз рад су уведена три значајно различита програмска језика. Сваки од њих има своје предности и мане, и у зависности од потребе треба проценити који приступ ће најбрже дати најпрецизније резултате. Ово уопштено важи за програмирање као дисциплину: као што добар мајстор неће чекићем шрафити шраф, тако и добар програмер треба да зна да процени који је најбољи алат за дати посао.

Најважније, циљ овог рада је био приказати колико информација нам је заправо на располагању. Интернет је неисцрпно добро, тако да чак и када администрација закаже при правилном отварању података, док год они постоје у било којој форми они нису „заробљени". Концептуално једноставни програми могу бити спона између *de jure* јавних информација и *de facto* отворених података и тиме отворити многа наизглед запечаћена врата.

Додатак А

Десктоп клијент –ако будем имао вишка времена

Додатак Б

Садржај базе података

Додатак B

Изворни код

Библиографија

[1] Портал отворених података. О порталу отворених података, 1.2.2018. URL https://data.gov.rs/sr/discover/.

[2] Geoff Boeing and Paul Waddell. New insights into rental housing markets across the united states: web scraping and analyzing craigslist rental listings. *Journal of Planning Education and Research*, 37(4):457–476, 2017.

[3] Suhit Gupta, Gail Kaiser, David Neistadt, and Peter Grimm. Dom-based content extraction of html documents. In *Proceedings of the 12th international conference on World Wide Web*, pages 207–214. ACM, 2003.

[4] W3 Конзорцијум. Document Object Model (DOM), Јануар 2005. URL https://www.w3.org/DOM/#what.

[5] Република Србија. Закон о ауторским и сродним правима. *Службени гласник Републике Србије*, (104/2009, 99/2011, 119/2012 и 29/2016 - одлука УС), 2009-2016.

[6] Република Србија. Закон о слободном приступу информацијама од јавног значаја. *Службени гласник Републике Србије*, (120/2004, 54/2007, 104/2009 и 36/2010), 2004-2010.

[7] Web Robot Pages. Robots exclusion standard, 1.2.2018. URL http://www.robotstxt.org/orig.html.

[8] Министарство просвете, науке и технолошког развоја. Упис у средње школе Републике Србије, 1.2.2018. URL http://upis.mpn.gov.rs/.

[9] Jordan S. Gruber. Outta site. *Wired*, Фебруар 1997. URL https://www.wired.com/1997/02/outta-site/.

[10] Apache HTTP Server Project. Performance scaling, 3.2.2017. URL https://httpd.apache.org/docs/trunk/misc/perf-scaling.html.

[11] Martin Abadi and Luca Cardelli. *A theory of objects*. Springer Science & Business Media, 1996.

[12] Jeffrey M. Barnes. Object-relational mapping as a persistence mechanism for object-oriented applications. *Mathematics, Statistics, and Computer Science Honors Projects*, 6, 2007.

[13] Завод за вредновање квалитета образовања и васпитања. Извештај о реализацији и резултатима завршног испита на крају основног образовања и васпитања у школској 2014/2015. години. 2015.

[14] Завод за вредновање квалитета образовања и васпитања. Извештај о резултатима завршног испита ученика припадника националних мањина -школска 2015/2016. година-. 2016.

[15] Завод за вредновање квалитета образовања и васпитања. Извештај о резултатима завршног испита ученика припадника националних мањина за школску 2016/17. годину. 2017.