Hijerarhijski i relacijski modeli kao bibliografkse baze podataka

Martinović, Marijan

Sveučilište u Zadru

Odjel za informacijske znanosti
Preddiplomski sveučilišni studij informacijskih znanosti

Marijan Martinović

Hijerarhijski i relacijski modeli kao bibliografkse baze podataka

Završni rad

Zadar, 2020.

Izjava o akademskoj čestitosti

Ja, **Marijan Martinović,** ovime izjavljujem da je moj završni rad pod naslovom **Hijerarhijski i relacijski modeli kao bibliografske baze podataka** rezultat mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na izvore i radove navedene u bilješkama i popisu literature. Ni jedan dio mojega rada nije napisan na nedopušten način, odnosno nije prepisan iz necitiranih radova i ne krši bilo čija autorska prava.

Izjavljujem da ni jedan dio ovoga rada nije iskorišten u kojem drugom radu pri bilo kojoj drugoj visokoškolskoj, znanstvenoj, obrazovnoj ili inoj ustanovi.

Sadržaj mojega rada u potpunosti odgovara sadržaju obranjenoga i nakon obrane uređenoga rada.

Zadar, 20. siječanj 2020.

Sažetak

Glavni cilj ovoga rada je uspoređivanje relacijske baze podataka koje koriste SQL jezik, i neralacijske baze podataka MongoDB koja koristi JSON format. Rad istražuje način zapisivanja bibliografskih metapodataka u internacionalnim standardima ISBD i MARC. Te uspoređuje njihovu strukturu sa strukturama baze podataka. Dok je bibliografska kontrola stara tema knjižnične rasprava, te su određeni standardi odavno odlučeni za određena područja, poput ISBD-a za kataložnu i knjižničnu svrhu, te zatim MARC koji je srodan ISBD strukturi, za računalnu djelatnost. U drugu ruku relacijske baze podataka su popularne od početka prošlog stoljeća, dok su neralacijske počele sa aktivnim reklamiranjem prošlom desetljeću, još nema priznatog standarda za bibliografske podatke. SQL jezik je začet na ideji relacijskih podataka, što bi značilo da tablica sadrži kodirane podatke koji vode do drugih povezanih tablica u kojima su upisane prave vrijednosti, što je srodno MARC formatu koji također ima kodirane podatke. U drugu ruku nerelacijske baze podataka dopuštaju hijerarhičku strukturu podataka što im dopušta veću fleksibilnost. Dok su obe strukture podataka idealne na svoje različite načine, oboje imaju i svoje različite mane. Relacijske baze inzistiraju na standardiziranim podacima te ograničuju unos, dok nerelacijske nemaju nikakve standarde osim onih zadanih od strane programera. Cilj rada je doći do kojekakvog idealnog rješenja i pronaći strukturu koja bi najbolje odgovarala bibliografskim metapodacima.

Ključne riječi: ISBD, MARC, UNIMARC, SQL, NoSQL, MongoDB, JSON,BSON,FRBR, BSON, bibliografski metapodaci, podaci, baza podataka

Sadržaj

Uvod

Pohranjivanje i organizacija podataka postoji još od početka pisma te se dan danas razvija i evoluira. Bibliografski metapodatci su glavni dio organizacije građe u informacijskim ustanovama. Pojava računala i strojno čitljivih formata dovela je do razvoja i rasta popularnosti baza podataka te njihovih publika. Bibliografski metapodatci se tako danas pohranjuju u baze podataka. Struktura u kojuj su metapodatci spremljeni su ključni za funkcionalnost baze pri spremanju, pohranjivanju i dohvaćanju podataka.

Baze podataka se danas često koriste za spremanje i organiziranje velikih količina podataka. Uz pomoć računala pohrana velikih količina podataka nije problem fizičkog prostora koliko je problem pravilnog strukturiranja podataka tako da podaci postoje u strojno čitljivom, ali i dobro strukturiranom i široko iskoristivom obliku. Važnost strukturiranih podataka leži u mogućnosti razmjene podataka između ustanova, kvalitetnoj kontroli podataka i pohrani podataka, što nas dovodi do strandardizacije te razvijanje formata za strojno čitanje i obradu podataka.

Uz razvijanje računalnih tehnologija, knjižnjičari razvijaju MARC strojno čitljive formate koji se unaprijeđuju i razvijaju dan danas. UNIMARC je standard koji je nastao 1973. godine te se smatra standardnim MARC formatom za knjižnične potrebe. Nastao je nakon broja raznih MARC standarda poput, UKMARC, USMARC, CAN/MARC, MARC21. MARC 21 je korišten u Kongresnoj knjižnici, a nastao je iz primjera anglo-američkog kataložnog listića i knjižničnih pravila. Format je nastao kao kombinacijom kanadskog i američkog MARC formata (USMARC i CAN/MARC). Dok su prijašnji MARC formati stvoreni na osnovama knjižničnih pravila pojedinačne države, UNIMARC je stvoren na osnovi iskustva prijašnjih MARC formata te uz pomoć međunarodne kataložne zajednice (UBC: akronim od engl. Universal Bibliographic Control) i međunarodne federacije knjižničarskih društava i ustanova (IFLA: akronim od engl. International Federation of Library Associations and Institutions). Kao rezultat UNIMARC je veoma stručno definiran format za knjižničnu razmjenu i pohranu, što omugoćava jasno definiranu strukturu podataka.

MARC formati su nastali kao strukturirani strojno čitljiv format. Otkada su nastali MARC formati, strukturirani podaci se pohranjaju u računalne baze podataka. Baze podataka su ključni dio organizacijske strukture, ovaj rad ispituje mogućnosti hijerarhijskih baza i relacijskih baza u bibliografske svrhe. Postoji više modela prema kojima su podaci strukturirani u bazi. Hijerarhijska baza uz pomoć svojeg modela koji dopušta širenje svojih

glavnih pojmova, u manje, više specifična pojmove, dopušta korisniku način povezivanja specifičnih atributa sa više abstraktom idejom. Uz pomoć ovakvih podataka, možemo koristiti veliki broj različitih atributa bez puno truda ili regulacije od strane baze. Dok je regulacija podržana, zahtjeva od korisnika da odredi regulacije za svaki novi atribut.

Relacijska baza podataka se sastoji od tablica koje se međusobno povezuju, dok svaka tablica drži jedinstvenu informaciju o tom jednom entitetu. Pretraživanje podataka zahtjeva veliku kontrolu nad samom bazom, podrazumjevajući da je sama baza pravilo zapisana. Također, relacijske baze podataka pretpostavlja regulaciju podataka u samoj definiciji baze. Popularan primjer softverski sustava koji je primarno hijerarhijski orijentiran je MongoDB, a relacijskog sustava PostgreSQL. Kroz ovaj rad istražujemo koristi i nedostatke obe baze podataka u bibliografske svrhe.

Za potrebe ovog rada podatke za bazu uzimamo iz Open Library-a uz pomoć API pristupa, te pratimo mogućnosti dvije baze podataka pri zapisivanju istih informacija. Open Library je projekt koji je nastao sa svrhom prikupljanja bibliografskih podataka svih knjiga ikad izdanih, što ga uz njegovo besplatno korištenje, čini ga idealnim za korištenje u ovom radu. Pri razmjeni podataka preko interneta sami podaci često nisu rezultat brojnih knjižničnih standarda i kontrola. Za svrhu ovog rada cilj nije ispisati sve ponuđene informacije i razvijati kompletan model kontrole bibliografskih informacija nego zapisati desetak ključnih za identifikaciju građe kako bi se ispitali različiti pristupi pohrani tih podataka.

Bibliografski standardi

ISBD

ISBD stoji za International Standard Bibliographic Description [1], ili na hrvatskom, Međunarodni standardni bibliografski opis. ISBD se sastoji od skupa pravila i skupina uz pomoć kojih se standarizira bibliografski opis djela za lakšu razmjenu opisa publikacije. ISBD je smišljen od strane IFLA-e (International Federation of Library Associations and Institutions). Postoji više izdanja ISBD-a od kojih se svaki bavi drugačijom građom. Struktura ISBD-a se sastoji od devet skupina, svaka od kojih ima svoje hijerarhijske podskupine. Uz pomoć ovih devet skupina opisujemo djelo, skupine su:

-0. Oblik sadržaja i vrsta medija

-1. Stvarni naslov i podaci o odgovornosti

-2. Izdanje

-3. Podaci specifični za građu

-4. Izdavanje, raspačavanje

-5. Materijalni opis

-6. Nakladnička cjelina

-7. Napomena

-8. Standardni broj

Primjer zapisa:

ROTHFUSS, Patrick J.

Ime vjetra : prvi dan / Patrick Rothfuss ; [prevela s engleskoga Petra Mrduljaš Doležal]. - Zagreb : Algoritam, 2012. - 723 str. : ilustr. ; 25 cm. - (Kronike Kraljosjeka;knj.1) Prijevod djela: The name of the wind. ISBN 978-953-316-375-8

[1] Verona, E. Međunarodni ured za univerzalnu bibliografsku kontrolu (International Office for UBC). // Vjesnik bibliotekara Hrvatske 20(1974)1-4, 50-52..

Gledajući kako je ISBD izvorno stvoren za analogne medije poput knjižničnih kataloga i bibliografija, dolazi do problema pri interoperabilnosti sa digitalnim medijima. U svrhu tog problema razvija se MARC strojno čitljivi standard. U ovom radu koristimo usluge Open Library projeka koji nam pruža mogućnost pretraživanja knjiga po njihovom ISBD-u.

MARC

Knjižnični format za strojno čitanje je MARC format. Stanard koristi sustav polja, blokova i indikatora. Standard dopušta veliku razinu preciznosti pri opisu građe. MARC stoji za Machine Readable Cataloging[2], ili na hrvatskom, strojno čitljiv katalog. Standard je počeo 1960. u Kongresnoj knjižnici sa svrhom da se stvori standard koji je namjenjen prvenstveno za strojnu razmjenu i pohranu kataložnih podataka. Naziv prvog formata je tada bio MARC I. Nakon velikog uspjeha MARC I formata, razvija se i sljedeći MARC II. Potražnja za standardom raste sljedećih godina te format koji je počeo u Kongresnoj knjižnici uzima ima USMARC. Ovaj format zauzima mjesto glavno formata u SAD-u i Kanadi. Uskladno s rastom i razvojem USMARC-a, razvija se i UKMARC u Britaniji. Početkom 1990. u svijetu je postojalo preko 50 različitih MARC formata, a svi su se bazirali na USMARC-u i UKMARC-u. Samim krajem 20-og stoljeća nastaje MARC 21 kao MARC format za 21 stoljeće. Ovaj format ujednačava različite inačice ranijih MARC formata.

[2] Willer, Mirna. UNIMARC u teoriji i praksi. Rijeka : Naklada Benja, 1996. Poglavlje 1 Formati za strojno čitljivo katalogiziranje i 2.1-2.6 UNIMARC i standardizacija.

UNIMARC

Struktura MARC formata se razvijala tijekom godina te su u uporabi mnoge inačice standarda, dan danas najkorišteniji međunarodni MARC standard je MARC 21, dok je UNIMARC najviše korišten u Europi. UNIMARC[3] ima sljedeću strukturu:

0-- Blok za identifikaciju

1-- Blok kodiranih informacija

2-- Blok glavnog opisa

3-- Blok napomena

4-- Blok za povezivanje kataložnih jedinica

5-- Blok srodnih naslova

6-- Blok sadržajne analize

7-- Blok podataka o odgovornosti

8-- Blok za međunarodnu upotrebu

UNIMARC čini veliku razliku od ISBD jer zanemaruje ISBD-ovu namjenu za kataložne listiće, već polja podataka strukturira prema njihovoj funkciji. Usprkos tome, MARC posjeduje broj polja koja su istovječna sa ISBD-om, samo donekle promještena u hijerarhiji i odnosima skupina. MARC formati su i nastali kako bi se iz njih nedvojbeno mogao stvoriti bibliografski zapis prema standardima nastalima za kataložne listiće, ali koji može pratiti i dodatne informacije. Veliki pomak pri strojoj čitljivosti čine kodirani podaci. Uz novi sustav blokova, također se koriste i indikatori i podpolja za definiciju bloka. Gledajući interakciju blokova i polja, možemo zaključiti da je MARC hijerarski organiziran standard.

[3] Willer, Mirna. UNIMARC u teoriji i praksi. Rijeka : Naklada Benja, 1996. Poglavlje 1 Formati za strojno čitljivo katalogiziranje i 2.1-2.6 UNIMARC i standardizacija.

Primjer MARC zapisa:

```
001    4320001587
010 #  #a978-953-316-375-8
100 #  #a20110325d2012 mmma0hrv01010101ba
101 0  #ahrv
       #ceng
200 1  #aIme vjetra
       #eprvidan
       #f Patrick Rothfuss
       #g[prevela s engleskoga Petra Mrduljaš Doležal]
210 #  #aZagreb
       #cAlgoritam
       #d2012
215 #  #a 723 str.
       #cilustr.
       #d25 cm
225 2  #aKronike Kraljosjeka
       #vknj. 1
300 #  #aPrijevod djela: The name of the wind
410 #  1 1 001432001586 (Kronike Kraljosjeka [Nakladnički niz])
675 #  #a821.111(73)-3
       #c Američka književnost. Romani, novele, pripovijetke
700 #  13 001431001537 (Rothfuss, Patrick J.)
702 #  13 0014017211 (Mrduljaš, Petra)
       4730a
999 # #91
```

MARCXML

Kroz godine MARC se nastavlja razvijati, pojavljuje se MARCXML, shema koja koristi MARC21 format i XML jezik za označivanje polja. XML (eXtensible Markup Language) je računalni jezik stvoren za očuvanje i dijeljenje podataka. XML je veoma sličan HTML(Hypertext Markup Language) jeziku koji se koristi za web stranice. MARCXML je nastao da iskoristi fleksibilnost XML jezika , dopušta veću razinu interoperabilnosti između

standarda. Konverzija iz MARC21 u MARCXML se vrši bez gubitka podataka, programi za konverziju su dostupni od strane Kongresne knjižnice.

Djelomični primjer MARCXML zapisa:

```
<?xml version="1.0" encoding="UTF-8"?>
<record
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.loc.gov/MARC21/slim
http://www.loc.gov/standards/marcxml/schema/MARC21slim.xsd"
    xmlns="http://www.loc.gov/MARC21/slim">

  <leader>     nam a22     7i 4500</leader>
  <datafield tag="999" ind1=" " ind2=" ">
    <subfield code="c">385234</subfield>
    <subfield code="d">385231</subfield>
  </datafield>
  <controlfield tag="003">HR-ZaNSK</controlfield>
  <controlfield tag="005">20180619133248.0</controlfield>
  <controlfield tag="008">050224s1994    xxub          000 f eng</controlfield>
  <datafield tag="020" ind1=" " ind2=" ">
    <subfield code="a">0618153985</subfield>
  </datafield>
  <datafield tag="035" ind1=" " ind2=" ">
    <subfield code="a">(HR-ZaNSK)000552123</subfield>
  </datafield>
  <datafield tag="040" ind1=" " ind2=" ">
    <subfield code="a">HR-ZaNSK </subfield>
    <subfield code="b">hrv </subfield>
    <subfield code="c">HR-ZaNSK </subfield>
    <subfield code="e">ppiak</subfield>
    <subfield code="d">HR-ZaFF</subfield>
  </datafield>
…
</record>
```

MODS

Mods stoji za Metadata Object Description Schema, shema koja koristi XML jezik za zapis podataka. Slično MARCXML dok ima veliku razliku korištenja blokova na bazi jezika, a ne kodova kao MARC standard.

Djelomični primjer MODS zapisa:

```
<?xml version="1.0" encoding="UTF-8"?>
<mods                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.loc.gov/mods/v3"                              version="3.1"
xsi:schemaLocation="http://www.loc.gov/mods/v3
http://www.loc.gov/standards/mods/v3/mods-3-1.xsd">
  <titleInfo>
    <nonSort>The </nonSort>
    <title>fellowship of the ring</title>
    <subTitle>being the first part of The lord of the ring</subTitle>
  </titleInfo>
  <name type="personal">
    <namePart>Tolkien, John Ronald Reuel</namePart>
    <role>
      <roleTerm authority="marcrelator" type="text">creator</roleTerm>
    </role>
  </name>
  <typeOfResource>text</typeOfResource>
  <genre authority="marc">novel</genre>
  <originInfo>
    <place>
      <placeTerm type="code" authority="marccountry">xxu</placeTerm>
    </place>
    <place>
      <placeTerm type="text">Boston</placeTerm>
    </place>
    <place>
      <placeTerm type="text">New York</placeTerm>
    </place>
    <publisher>Houghton Mifflin Company</publisher>
    <dateIssued>cop. 1994</dateIssued>
    <dateIssued encoding="marc">1994</dateIssued>
  </recordInfo>
</mods
```

Korištenje XML strukture, hijerarhijsku strukturu i jezika za imenovanje polja je trend koji se nastavlja dan danas kada je JSON format glavni format za razmjenu podataka. Smjer u kojem formati za ramjenu idu se bazira na jednostavnosti i čitljivosti.

Struktura i modeli podataka

Svi podaci organizirani za računalnu iskoristivost su strukturirani prema nekom modelu. Strukture podataka u knjižnici su doživjele mnoge promjene uz napredak računalnih tehnologija. MARC format je veoma koristan unutar poslovanja informacijske ustanove, često je glavni format za razmjenu podataka radi njegovog usporednog razvoja sa ISBD-om.
XML i JSON formati su puno češći format za razmjenu podataka u raznim okruženjima, formati dopuštaju korisniku imenovanje atributa. Također su formati namješteni u veoma slobodnu hijerarhijsku strukturu, koje daju korisniku slobodu grananja atributa. Hijerarhijska struktura podataka i relacijska struktura podataka su dva naj češća modela za razmjenu podataka.

Tehnologije

Relacijski podatci i relacijska baza podatka

Relacijska baza podatka znači da se svi podatci skupljaju i definiraju na osnovu relacija između podataka. Relacijske baze podataka su se pojavile u upotrebi sedamdesetih godina prošlog stoljeća. Osnivač ovakvog sustava se smatra Edgar Codd, direktor IBM-ovog sektora za proizvodnju računalnih tvrdih diskova. Tijekom prošlog stoljeća prve baze podataka su bile osmišljene za suvremenu računalnu memoriju poput tvrdih diskova i programerski pristup podacima su koristili hijerarhijski model. Takav sustav se smatrao neefikasnim prvenstveno zbog nedostataka hijerhijskog sustava, koji stvara veliki broj dupliciranih podataka i prouzročava problem memorije. Radi tih razloga Edgar Codd[4] godine 1970. objavljuje svoj rad o relacijskim bazama podataka pod imenom „Relacijski model za organizaciju velikih količina podataka“(„A Relational Model of Dana for Large Shared Dana Banks“). U radu objašnjava prednosti relacijskog modela preko hijerarhijskog radi mogućnosti stavljanja podatka u zasebne tablice umjesto u povezane liste navigacijskog sustava. Današnje relacijske baze se koriste istim sustavom dvodimenzionalnih tablica za pohranu podataka. Podatke se zapisuje u tablice te se svaka sastoji od stupaca i redaka. Za svaki redak u tablici je

4 Kemić, Želimir. (2013) *Primjer baze podataka u sustavu MySQL.* Završni rad. Varaždin: Sveučilište u Zagrebu , Fakultet organizacije i informatike,

entitet koji stoji za bilo koju stvar, osobu, riječ ili pojavu, a naredni stupci opisuju atribute tog entiteta.

Primjer : tablica iz relacijske baze podataka:

ID	Naslov	Izdavač	Godina objave
1	The Kingkiller Chronicle: The Name of the Wind	DAW Books	2007.
2	The Kingkiller Chronicle: The Wise Man's Fear	DAW Books	2011.
3	The Way of Kings	Tor Books	2010.

Primarni ključ je atribut koji jedinstveno opisuje svaki od nabrojenih entiteta. Jednostavnije rečeno, to je jedinstvena šifra nekog individualnog zapisa. Primarni ključ nam služi za povezivanje s drugim tablicama. Kada primarni ključ neke tablice koristimo u drugoj tablici kako bi ostvarili vezu, tada taj ključ u drugoj tablici nazivamo strani ključ. U navedenom primjeru vidimo da se pod atribute „Izdavač“ pojavljuje dvaput isti atribut „DAW Books“. U navedenom primjeru ovakav problem se ne čini problematičnim, ali kod velikog broja entiteta jedan autor može sadržavati veliki broj djela, što bi beskorisno zadržavalo mjesto u tablici, imajući jedno ime da se ponavlja više desetaka puta. Stoga da bi se izbjeglo gomilanje bespotrebnih podataka prilikom kreairanja baze podataka potrebno je napraviti odvojenu tablicu za autore čiji primarnu ključevi („ID“) će se vezati na naš atribut autora, te s tim postaje strani ključ tablice „Naslov“.

Primjer : relacije između tablica

ID	Naslov	ID_Izdavač	Godina
1	The Kingkiller Chronicle: The Name of the Wind	1	2007.
2	The Kingkiller Chronicle: The Wise Man's Fear	1	2011.
3	The Way of Kings	2	2010.

ID_Izdavač	Izdavač
1	DAW Books
2	Tor Books

U primjeru je navedena veza između te dvije tablice, odnosno njihova relacija, a po tipu i broju veza dva entiteta mogu imati više različitih relacija, koje sve mogu biti jedne od tri vrste relacija poput:

- 1:1 – relacija jedan naprama jedan, što bi značilo da za svaki jedan zapis u dvodimenzionalnoj tablici vezan samo jedan zapis u nekoj drugoj tablici. Ne koristi se odviše često jer takvi podaci mogu biti upisani svi na jednoj te istoj tablici. Najčešća upotreba ove veze je za optimizaciju operacija s bazom.
- 1:N – relacija jedan naprama više, najčešće je upotrebava veza među tablicama i znači da na jedan zapis u prvotnoj tablici može odnositi jedan ili više zapisa u nekog drugoj tablici.
- M:N – veza koja sadrži više atributa prema više, za slučajeve kada jedan i više entiteta bude povezano s jednim ili više entiteta u drugoj tablici. Nažalost ovakve veze nisu dopuštene u oblikovanju relacijske baze podataka, što se rješava uvođenjem treće tablice zvane tablica sjecišta, koja se veže u vezu 1:M s ostale dvije tablice. Na primjer jedan autor može imati više knjiga i jedna knjiga može imati više autora.

FRBR

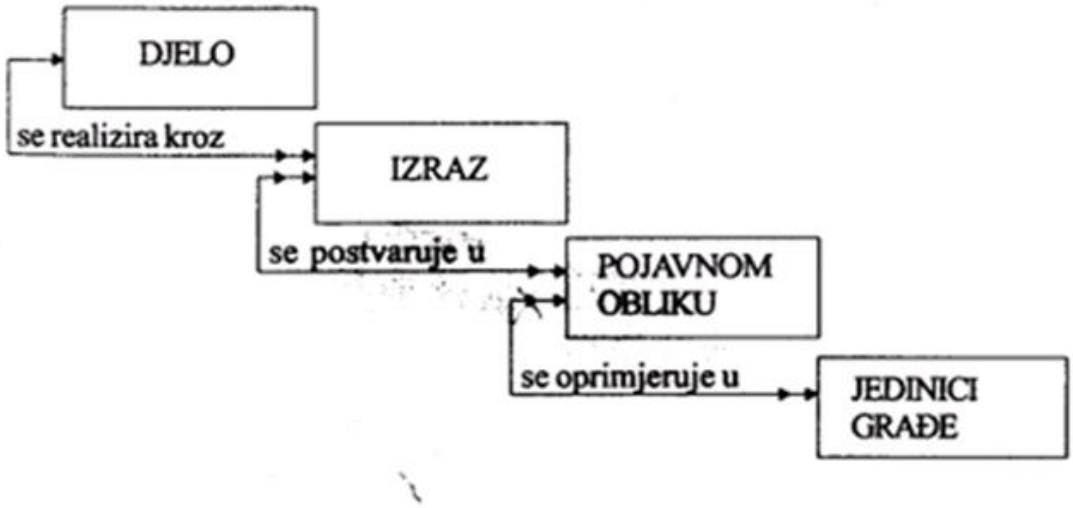
FRBR[5] je sustav relacijskih odnosa između bibliografskih podataka koji je smišljen kao novi pristup odnosu knjiga-katalog-korisnik. Struktura upita se ne promatra više linearno, kao u knjižničnom katalogu, već modularno, što je u skladu sa konceptualnim modelom za sustave relacijskih baza podataka, tj. samu strukturu entiteta i odnosa u okviru kojem se definiraju

- Entiteti: predmet zanimanja korisnika(npr. autor, naslov) – shvaća se kao središnja točka za skup podataka
- Odnosi: koji vladaju između entiteta – npr. odnos autor i djelo
- Atributi: svojstva svakog entiteta služe korisnicima pri oblikovanju upita(npr. usporedni stvarni naslov, podaci o odgovornosti, godina izdavanj)

FRBR nam pomaže pri pretrazi kada tražimo specifičan rad, između više derivacija istog djela naći onaj koji nam je potreban.Npr. kada tražimo predstavu Hamleta, prvo će nam izaći djelo Williama Shakespeara, zatim derivacije i drugačije verzije istog djela, te djela koja se bave djelom Hamlet, predstava neće biti prvi rezultat, ali FRBR nam pomaže pri pronalasku djela.

[5] Denton, W. FRBR and the history of cataloging. // Understanding FRBR: what it is and how it will affect our retrieval tools / edited by A. G. Taylor. Westport, CT : Libraries Unlimited, 2007. Str. 35-57.

Vizualizirani koncept:



SQL programski jezik

SQL je skraćeno od Structured Query Language[6] . Stoji među najpoznatijim računalnim jezicima za strukturiranje podataka, manipulacijom i održavanjem baze podataka. SQL kao računalski jezik je napravila i razvila američka tvrtka IBM. U početku je jezik imao ime "System R", te je tadašnji zadatak tada zvanog Structured English Query Language("SEQUEL") bio upravljanje bazom podatka te njezinom manipulacijom. Kasnije u njegovom životu kratica SEQUEL je promjenjena u SQL radi pravnih razloga. Dan danas SQL je veoma popularan kao jezik za manipulaciju radi njegovog laganog pristupa za korištenje. Važno je spomenuti da je SQL deklarativan jezik, odnosno korisnik formalno govori *što* se treba provesti, a softverskom sustavu kojim je implementirana baza je prepušteno da odluči *kako.* Ovo ga čini podobnim za upravljanje podacima. To ga čini različitim od najčešće paradigme u programskim jezicima koja je imperativna, odnosno korisnik piše *kako* će se neki proces izvesti. Iz aspekta upravljanja podacima, ovo umanjuje

[6] Kemić, Želimir. (2013) *Primjer baze podataka u sustavu MySQL.* Završni rad. Varaždin: Sveučilište u Zagrebu , Fakultet organizacije i informatike,

mogućnost pogreške i pospješuje efikasnost. Danas sve relacijske baze podataka podržavaju SQL. Neki primjeri popularnih sustava su:

- MySQL
- PostgreSQL
- SQLite
- Oracle
- Microsoft SQL Server

SQLite

Od čestih inačica SQL koda i jezika, SQLite je jedan od najpoznatijih i najnaprednijih relacijskih sustava otvorenog koda za pristup i korištenje baze podataka. SQLite je program otvorenog koda i besplatan za većinu korisnika. Sustav je veoma dobro optimiziran te se pokreće sa mnoštva različitih operacijskih sustava kao što su MacOS, Windows i Linux. Radi njegove stabilnosti Linux je najčešći operacijski sustav za uspostavu servera i baza podataka. SQLite ne zahtjeva svoj klient već se može koristiti uz pomoć priključka unutar Python koda ili nekog drugog programskog jezika.
Za razliku od ostalih ovdje spomenutih sustava, značajno je spomenuti da je SQLite, kao što ime govori, namijenjen kao mala baza podataka koja se direktno ugrađuje u neki softver, a ne kao zaseban sustav. Ipak, upravo ga ova značajka čini podobnim za prikazivanje mogućnosti relacijskog modela, ali valja držati na umu da SQLite nije podoban za kataloge većih ustanova.

SQLite Viewer with Google Drive

SQLite Viewer with Google Drive je priključna aplikacija za Google Chrome web pretraživač koja omogućuje pregled SQL baza i datoteka. Gledajući kako SQLite nema svoje grafičko sučelje, uz pomoć ovog dodatka na Google Chrome možemo pratiti progres naše relacijske baze. Dapače, striktno govoreći niti jedna baza podataka nema grafičko sučelje sama po sebi jer je namjena posluživati podatke programima, grafičko sučelje služi samo upravljanju bazom. Ipak, mnoge tvrtke koje razvijaju baze podataka, razvijaju i grafička sučelja za upravljanje njima, ali ista je važno razlikovati od samih baza.

Hijerarhijski podaci i nerelacijska baza podataka

Uz pomoć JSON formatu podataka, moguće je podatke staviti u hijerarhiju, takozvani ugnježđeni podaci. Uz pomoć ove opcije možemo staviti više podataka pod jednu skupinu, kao na primjer sa autorima:

Primjer hijerarhijskog zapisa:

```
Zapis = {
      „naslov“:“preddiplomski studij“
      „autor“:[
      {„Ime“:“Ana“, „Prezime“:“Anić“},
      {„Ime“:“Ivo“,“Prezime“:“Ivić“}
      ]}
```

Primjer demonstrira kako u vrijednost „autor“ možemo pohraniti više autora, što je veoma korisno kod djela koja ima više autora, također istu strukturu možemo upotrijebiti na ostalim atributima, recimo kada imamo više od jednog izdavača.

Važno je napomenuti da MongoDB i ostale polustrukturirane i nerelacijske baze podataka nemaju definiranu shemu. Unutar organiziranja relacijske baze podataka, treba točno definirati strukturu podataka i njihove relacije. Što znači da u takvim bazama (prvenstveno SQL bazama) za svaki podatak treba biti definirana tablica, broj kolumni, redova, vrsta podataka i duljina za svako polje u tablici, te uz to svako polje mora prolaziti određene mu kriterije.

MongoDB

MongoDB je baza podataka NoSQL baza podataka otvorenog koda. NoSQL baze podataka se ne koriste višestrukim povezanim tablicama radi tvorenja veza između tablica, već sve podatke unosi na svoju shemu, MongoDB koristi JSON[7] oblik informacija zvan BSON.. Kako Mongo koristi JSON format podataka što bi značilo da se podaci skladište u strukturu koja se u mnogim programskim jezicima naziva „rječnik“. Jednostavno rečeno rječnik je skup ključ-vrijednost pareva gdje je ključ naziv atributa (i.e. „polja“), a vrijednost je vrijednost tog atributa, a kod SQL sustava, „ključ“ je šifra zapisa koji se sastoji od više atributa i njihovih vrijednosti. U svakom slučaju, činjenica da se zasniva na JSON pristupu podacima daje ovom formatu veliku fleksibilnost i mogućnost pri pohrani i manipuliranju u računalnom okružju.

Velika prednost mu je da je u direktnom odnosu sa strukturama podataka koje se često koriste u programskim jezicima. Drugim riječima, za razliku od npr. XML-a i relacijskih podataka, podatke nije potrebno prvo transformirati u strukture podobne za programiranje već su podaci tako i zapisani.

Glavna razdvojna točna Mongo i SQL sustava je ta da Mongo ne zahtjeva točnu shemu podataka te nudi fleksibilnu opciju skladištenja informacija. Jedine standarde koje Mongo zahjeva su oni koje mu programer koji se njemu trenutačno služi zahtjeva. Ako želimo striktnu kontrolu podataka možemo je ugraditi, ali sustav ju ne zahtijeva.

Prvotna zamisao MongoDB-a bila kao baza podataka opće namjene, ima puno opcija koje također uključuju:

- Indeksiranje – omogućava brzo izvršavanje različitih akcija, te omogućuje jedinstvenu, spoj i cijelog teksta vrste indeksiranja
- Agregacije – daju nam opciju izrađivanja složenih zahtjeva od više manjih zahtjeva
- Posebne vrste zbirki – omoguće nam stvaranje zbirki, što su svojevrsne skupine unutar baze podataka, koje možemo odrediti raznim kriterijima, poput vrijeme trajanja ili veličine zbirke.
- Spremište podataka – omogućava jednostavan i prikladan način za spremanje veoma velikih skupova podataka.

[7] Peroković, Marko. (2014) *Primjena NoSQL baza podataka na sustav za upravljanje dokumentima*. Diplomski rad. Varaždin: Sveučilište u Zagrebu, Fakultet organizacije i informatike

Drugim riječima, MongoDB podržava sve što i veliki relacijski sustavi. Sama srž MongoDB-a su dokumenti: specifični zapisi koji se sastoje od ključeva koji su povezani sa vrijednostima. Prikaz podataka ovisi o samom programskom jeziku, ali velika većina jezika ima strukturu i metodologiju koja paše samoj strukturi JSON formata. Ključevi ili ID atributi su string vrsta zapisa, te je podržavan na svakoj UTF-8 kodnoj stranici, osim par iznimaka.

MongoDB – Zbirke

Zbirke (eng. „Collections“) su u suštini grupe dokumenata, veoma slično SQL tablicama, razni entiteti stavljeni u isti spremnik sa dodjeljenim ključevima ili ID vrijednostima.
Zbirke mogu imati različita ograničenja i sheme, na primjer jedna zbirka može primati drugačije vrste podataka od druge zbirke. Zbog tog razloga radimo više različitih zbirki, kao i zbog drugih razloga poput: lakše pretraživanje i snalaženje, brži uvid u dokumente sa traženim podacima, groupiranja po vrsti podataka. MongoDB također sam dodjeluje indekse svim zbirkama, tako da pri bazi s više zbirki istih vrsta podataka pronalaženje one jedne koje tražimo ne bude problem.
Identifikacija zbirke je stvar našeg imenovanja. Imenovati zbirku možemo sa bilo kojom kombinacijom UTF-8 kodiranog teksta, samo sa par iznimaka, kao što je prazan tekst, jer zbirka mora imati ime. Zbog istog razloga nije moguće staviti null vrijednosti u ime zbirke. Oba razloga imaju smisla dok treći je nemogućnost imenovanja zbirke sa prefiksom „system.“ jer je to određeno za interne zbirke u bazi podataka.
Također je moguće formirati pod-zbirke, to jest zbirke unutar zbirki koje se odvajaju točkom. Primjera radi moguće je napraviti zbirku knjiga zatim u njoj pod-zbirke nazvane knjiga.naslov i knjiga.autor, što bi pomoglo u organizacijske svrhe te pri dohvaćanju podataka.

MongoDB Compass Community

MongoDB Compass Community je GUI (graphical user interface), grafičko sučelje za MongoDB bazu podataka. Program je dostupan na Linux, Mac i Windows operativnim sustavima te je besplatan za sve korisnike. Korištenje grafičkog sučelja olakšava korištenje baze podataka te dopušta korisniku veću preglednost nad radom.

Python programski jezik

Python je programerski jezik osmišljen 1980. od strane Guido Von Rossuma u Nizozemskom Nacionalnom Istraživačkom Institutu za Matematiku i Računalne Znanosti (English: "National Research Institute for Mathematics and Computer Science"). Prva javna inačica jezika izlazi 1991. te kroz razvoj narednih godina do današnjice, Python postaje jedan od najpopularnijih programerskih jezika na svijetu odmah uz Javu i C. Među glavnim osobinama Python jezika je njegova fleksibilnost, što ga čini veoma popularnim programom za početnike i za iskusne programere. U ovom kontekstu, važno je napomenuti kako je Python vrlo popularan u poslovnoj i znanstvenoj zajednici kao jezik za upravljanje i analizu podataka.

PyCharm Edu

PyCharm Edu je besplatni program za programiranje u Pythonu. Osmišljen je da pomogne korisniku pri navigaciji i snalaženju u kodu. Program dopušta intergraciju s raznim dodatcima koji dopuštaju širenje sposobnosti programa. Uz pomoć dodataka za prihvaćanje URL API podataka, JSON formata, SQL priključka i MongoDB priključka, ovaj projekt obavljamo isključivo iz jednog programa, PyCharm Edu. Dok su ostale tehnologije poput SQLite Viewer i MongoDB Compass tu čisto radi pogodnosti grafičkog sučelja, PyCharm se koristi za izvršavanje svih procesa.

Open Library

Open Library je projekt koji cilja napraviti web stranicu za svaku knjigu ikad izdanu, web stranica ima preko 12 000 000 posjetitelja te oko 70 000 aktivnih korisnika. Open Library također pruža besplatnu API uslugu koja dopušta svakom korisniku pristup njihovim knjižnjičnim metapodatcima. Za razliku od Google Books API službe i Amazon API, Open Library pruža potpunu službu besplatno, dok puno ostalih usluga dopušta tek ograničen broj metapodataka, 500 po danu ili slično. Svrha ovog rada nije pronalazak idealnog API sustava već transformacija i bilježenje bibliografskih podataka u različitim bazama podataka, što čini Open Library uslugu idealnu za ovaj projekt.

Implementacija predloženog rješenja

Sljedeći dio rada se fokusira na implementaciji rješenja uz pomoć programskog jezika Python. Važno je napomenuti da navedena rješenja nisu isključivo jedina moguća točna rješenja, te da je moguće uz pomoć drugačijeg koda doći do istog ili različitog rješenja.

Dohvata podataka

Kod u ovom djelu se bavi dohvaćanjem podataka sa OpenLibrary i spremanjem u .json formatu.

Primjer dohvate i spremanja podataka:

```
1. import urllib.request
2. import json

3. url_pattern                                                                  =
   'https://openlibrary.org/api/books?bibkeys={}&jscmd=details&format=json'
4. prefix = "ISBN:"

5. isbns = [
6.         '9780515134483', '9780451190529', '9780312363055', '9780451210869',
7.         '9780385504201', '9780765356130', '9780345339737', '9781439133941',
8.         '9780786838653', '9780545060394', '9780345348104', '9781570086472',
9.         '9781416901945', '9780440227533', '9780439372978', '9780385732550',
10.        '9780553582017', '9780385490818', '9781593080273', '9780756404734',
11.        '9780688120498', '9780441005901'
12.]

13.parts = []
14.for isbn in isbns:
15.        parts.append(prefix + isbn)
16.url = url_pattern.format(','.join(parts))

17.response = urllib.request.urlopen(url)
18.content = response.read()
19.content = content.decode('utf-8')
20.response.close()
21.with open('./book_data.json', 'w', encoding='utf-8') as file:
22.        file.write(content)
```

Prva 2 redka koda zauzimaju moduli potrebni za ovaj zadatak. Urllib.request je modul koji se bavi dohvaćanjem podataka sa URL adrese. Json modul omogućuje čitanje i pisanje podataka u .json obliku.

Redak broj 3 i 4 zauzimaju komponete za poziv na API. Prvo „url_pattern“ čini dio URL adrese sa dodatkom „&jscmd=details&format=json“, koji definira razinu detalja i format u kojem su prikazani. Vitičaste zagrade {} čine placeholder, u koji će se različiti ISBNovi uvrštavati tako da „url_pattern“ čini jedinstveni URL za svaki ISBN koji imamo.

Redak broj 5 se sastoji samo od popisa svih ISBNova koje namjeravamo proći kroz kod, Svaki je uvršten u zagrade i odvojeni su zarezom.

Redak 13 do redka 16 sastoji se od djela za skidanje podataka. Uvrštena je petlja koja prolazi kroz sve ISBNove, jedan po jedan, te kroji jedinstveni „url_pattern“ za svaki ISBN.

Zadnji redci od 17 do 22 se bavi spremanjem podataka u datoteku „Book_data.json“.

Stvaranje SQLite baze

Sljedeći kod se fokusira na kreiranju prazne SQLite baze uz pomoć Pythona. Ovdje neće biti prikazan cijeli kod, tek dio koji se bavi definiranjem tablice za knjige,sličan postupak se ponavlja za stvaranje svih tablica.

Primjer stvaranja SQLite tablica:

```
Import sqlite3
table_names = ['BOOKS', 'AUTHORS', 'PUBLISHERS', 'BOOKAUTHOR']
drop_pattern = 'DROP TABLE IF EXISTS {};'


create_books = """
CREATE TABLE BOOKS (
       Book_id INTEGER PRIMARY KEY,
       Title TEXT NOT NULL,
       Subtitle TEXT,
       Series TEXT,
       ISBN10 TEXT,
       ISBN13 TEXT,
       Pages TEXT,
       Publish_date,
       Publisher_id INTEGER
);
"""
create_bookauthors = """
CREATE TABLE BOOKAUTHOR (
       Book_id INTEGER,
       Author_id INTEGER,
       PRIMARY KEY(Book_id, Author_id)
);
"""
```

Navedeni kod stvara dvije SQL tablice. Prvi redak zauzima modul sqlite3 koji nam dopušta spajanje na bazu i snimanje podataka.

Sljedeća dva redka dopuštaju stvaranje i brisanje tablica ako ih trebamo ponovo stvoriti.

Ostatak koda prikazuje definiranje dvaju tablica, tablica Books i tablica BookAuthor.

Tablica Books uključuje sve podatke koje stavljamo u tablicu, možemo ih vidjeti u prvom stupcu.

Drugi stupac ,napisan isključivo velikim slovima, nam služi za definiciju informacije koje atribut prima. Book_id atribut prima isključivo INTEGER koji nam služi kao PRIMARY KEY, glavni ključ koji nam služi za identifikaciju knjige. Ostali atributi primaju TEXT vrstu podataka.

Sljedeća tablica, BookAuthor, nam služi za spajanje tablica Book i tablica Author, sastoji se samo od dva atributa, Book_id i Author_id, čiju kombinaciju koristimo za spajanje knjiga i autora.

Na isti način stvaramo tablice za autore i izdavače, ali gledajući kako je proces veoma sličan ili isti, nisam ga uključio.

Primjer stvaranja SQLite baze:

```
create_tables = [create_books, create_publishers,
                 create_authors, create_bookauthors]
sqlite_file = './BiblioBaza.sqlite'
conn = sqlite3.connect(sqlite_file)
c = conn.cursor()
for name in table_names:
      c.execute(drop_pattern.format(name))
for create in create_tables:
      c.execute(create)
conn.commit()
```

Nakon definiranja svih tablica,stvaramo bazu, ovdje nazvanu BiblioBaza.sqlite.

Uz pomoć funkcije „conn=sqlite3.connect()“ koja dolazi iz sqlite3 modula, spajamo se na bazu te pokrećemo postupak brisanja baze ako ista već postoji sa funkcijom „drop_pattern.format()“. Tek nakon čišćenja, stvaramo novu bazu sa svim željenim tablicama sa funkcijom „create()“.

MongoDB baza i spremanje podataka

Za razliku od SQLite baze, MongoDB zahtjeva manje koraka za uspostaviti. Hijerarhijska priroda baze zahtjeva samo jednu bazu, sa jednom kolekcijom koju ćemo imenovati „Books“. Kako je samu bazu lakše uspostaviti, tako je i kod potreban za izvlačenje i spremanje podataka puno jednostavniji. Usprkos tome kod radi više različitih funkcija radi čega ga prikazujem u djelovima.

Primjer početka MongoDB koda:

```
import pymongo
import json

data_path = './book_data.json'
with open(data_path, encoding='utf-8') as file:
    data = json.load(file)

myclient = pymongo.MongoClient("mongodb://localhost:27017/Project")
mydb = myclient["Baza"]
mycol = mydb["Books"]
```

Kao i prije kod počinje sa modulima potrebnim za funkcionalnost koda. Pymongo za pristupanje i kontrolu nad MongoDB bazom i Json za mogućnosti Json formata.

Srednji dio primjera koda nam služi za učitavanje podataka, koji smo spremili sa prvim kodom za dohvaćanje i spremanje u .json formatu.

Zadnji dio primjera stvara MongoDB bazu uz pomoć pymongo modula, definira ime baze i ime kolekcije.

Primjer sredine MongoDB koda:

```
book_count = 0
for key in data:
    book_data = data[key]['details']
    book_count += 1
    book_id = book_count
    title = book_data['title']
    subtitle = book_data.get('subtitle')
    pages = book_data.get('number_of_pages')
    publish_date = book_data.get('publish_date')
    series = book_data.get('series')
    if series:
        series = ','.join(series).replace(';', '')
    isbn_10 = book_data.get('isbn_10')
    if isbn_10:
        isbn_10 = ','.join(isbn_10)
    isbn_13 = book_data.get('isbn_13')
    if isbn_13:
        isbn_13 = ','.join(isbn_13)

    publisher = book_data.get('publishers')
    if publisher:
        publisher = ';'.join(publisher)

    authors = data.get('authors')
    author_value = []
    if authors:
        for author_data in authors:
            author_value.append(author_data['name'])
```

Srednji dio koda počinje sa stvaranjem popisa „book_count=0“ koji nam služi za numeniranje svake knjige, u trećem redku vidimo „book_count +=1“ što nam povećava taj broj za 1 za svaku knjigu.

Petlja koja počinje na početku funkcionira na svakoj obrađenoj knjizi, te iz podataka dohvaća tražete podatke sa funkcijom „.get“. U par primjera podatke dodatno čistimo sa dodatnom petljom, kao npr. „series“. Petlja je u funkciji mijenjanja znakova koji se mogu dokazati problematični. U slučaju sa „isbn_10“ i „isbn_13“, mogućnost više od jednog ISBNa riješavamo sa petljom koja ih odvaja sa zarezom, isto u slučaju „publisher“.

Primjer zadnjeg dijela MongoDB koda:

```
    book_document = {
        "_id": book_id, "Title": title, "Subtitle": subtitle,
        "Series": series, "Author": author_value,
        "Publisher": publisher, "Publish date": publish_date,
        "ISBN10": isbn_10, "ISBN13": isbn_13,
        "Pages": pages
    }


    x = mycol.insert_one(book_document)
```

Zadnji dio koda se bavi spremanjem podataka. „Book_document“ uvrštava sve podatke koje smo izvukli iz skinutih podataka, i uvrštava ih pripadajućem atributu. Zadnja linija koda sprema zapis knjige u MongoDB bazu.

Spremanje podataka u SQLite bazu

Izvlačenje podataka iz .json datoteke za SQLite bazu je identičan kao za MongoDB. Prvi primjer prikazuje spajanje na bazu i definiranje vrijednosti koje ulaze u bazu.

Primjer početka SQLite koda:

```
sqlite_file = 'BiblioBaza.sqlite'
conn = sqlite3.connect(sqlite_file)
c = conn.cursor()

sql_aut = "INSERT INTO AUTHORS ('Author_id','Author') VALUES ({},'{}');"
sql_book_aut = "INSERT INTO BOOKAUTHOR ('Book_id','Author_id') VALUES ({},{});"
sql_pub = "INSERT INTO PUBLISHERS ('Publisher_id','Publisher','City') VALUES
({},'{}','{}');"
sql_book = """INSERT INTO BOOKS
('Book_id','Title','Subtitle','Series','ISBN10','ISBN13',
'Pages','Publish_date','Publisher_id')
VALUES ({},'{}','{}','{}','{}','{}','{}','{}',{});"""

book_count = 0
author_count = 0
publisher_count = 0

author_registry = {}
publisher_registry = {}
```

Prvi dio koda služi za pristupanje bazi 'Bibliobaza.sqlite'.

Srednji dio koda definira „sql_aut/book_aut/pub/book“, prve zagrade definiraju entitete u tablici, zatim druge zagrade koje sadrže prazne placeholdere služe za ubacivanje podataka u te pozicije. Ovaj koncept će biti naknadno pojašnjen u daljnjem dijelu koda.

Zadnji dio koda otvara prazne liste radi izbjegavanja duplih podataka, imamo „book/author/publisher_count“ intiger stavljen na 0 koji će se povećavati sa svakom knjigom/autorom/izdavačem obrađenim. Ostale dvije liste se primaju imena autora ili izdavača, tako da znamo ako smo se već susreli s njima, pojašnjeno u daljnjem kodu.

Primjer ostatka SQLite koda:

```
for key in data:
    book_data = data[key]['details']
    book_count += 1
    book_id = book_count
    title = book_data['title']
    subtitle = book_data.get('subtitle')
    series = book_data.get('series')
    if series:
        series = ','.join(series).replace(';', '')

    authors = data.get('authors')
    if authors:
        for author_data in authors:
            name = author_data['name']
            if name in author_registry:
                author_id = author_registry[name]
            else:
                author_count += 1
                author_id = author_count
                author_registry[name] = author_id

                c.execute(sql_aut.format(author_id, name))

            c.execute(sql_book_aut.format(book_id, author_id))


    bk_sql = sql_book.format(book_id, title, subtitle, series, isbn_10,
                             isbn_13, pages, publish_date, publisher_id)
    c.execute(bk_sql)
```

Radi ponavljajuće prirode koda, u zadnjem primjeru navedeno je tek par stavki koje vrijedi daljnje pojasniti.

Većinu podataka dohvatimo poprilično jednostavno uz „.get“ funkciju , određene entitete koju imaju mogućnost sadržavanja problematičnih znakova, filtriramo uz pomoć „.replace“ funkcije.

Za vrijednost autora moramo se pobrinuti da je svaki autor jedinstven i ne ponavlja se , inače nema razloga koristiti tabličnu bazu podataka. Kao što vidimo u kodu, stvaramo petlju koja prvo pronalazi autorovo ime u listi „author_registry“ koju smo formirali praznu u prijašnjem kodu. Ako je autor pronađen njegov „author_id“ je isti sa brojem autora u „author_registy“ te broj autora ostaje isti. Ako autor nije pronađen, on se stavlja u listu i dodan mu je novi broj.

Zatim ako je jedna od te dvije akcije izvedena, autor se stavlja u tablicu za autora i u tablicu za povezivanje autora i izdavača. Isti postupak se vrši nad drugim informacijama koje imaju mogućnost ponavljanja, kao izdavač, ali taj dio nije prikazan ovdje radi ponavljajuće prirode.

Zadnji dio koda uvrštava se prijašnje detalje knjige u tablicu za knjige.

Zaključak

Ovaj rad ispituje prirodu bibliografskih podataka te istražuje njihove mogućnosti pohranjivanja u različitim strukturama. Relacijske i hijerarhijske strukture podataka obe imaju svoje mane i prednosti. Prednosti spremanja podataka u relacijske baze dolazi do izražaja pri velikim količinama podataka koje se često mijenjaju, dopuštaju promjenu jednog podataka koji se zatim bilježi na svim povezanim podacima. Ključna razlika relacijskih baza je nedostatak duplih podataka, što također daje relacijskih bazi prednost pri zahtjevu memorije. Ova struktura čini relacijsku bazu veoma popularnom u profesijama sa velikim brojem podataka i redovitih promjena. Prednost hijerarhijske baze je mogućnost vezanja svih podataka za jedan objekt, što čini takvu bazu puno lakšu za koristiti. Svaki objekt u hijerarhijskoj bazi je čitavi objekt, dok u relacijskoj bazi je potrebno povući podatke sa broja raznih tablica. Nedostatak ove strukture je broj ponavljajućih podataka koji nemožemo regulirati. Sigurno broj ponavljanja ovisi o vrsti bibliografskih podataka koje stavljamo u bazi, ako je subjekt knjižnična beletristika, broj autora po radu će vjerovati ostati oko jedan po djelu, nekad možda više. Ako za subjekt uzimamo znanstvene radove, gdje po radu imamo 30-40 autora, koji se zatim ponavljaju u ostalim radovima, tu dolazimo do problema. Usprkos tome vjerujem kako je hijerarhijska struktura više idealna za bibliografsku građu. Opisuje

jedno djelo u cijelosti i dopušta stvaranje novih pod-kategorija za specifičnije radove. Idealno riješenje bi zahtjevalo implementaciju relacijskih podataka u često područja gdje se često ponavlja ista informacija, kao izdavač ili autor. Nasreću većina baza podataka u upotrebi danas su hibridne baze, koje dopuštaju upravo to, spajanje više struktura podataka. Baze koje u osmišljene kao NoSQL baze poput MongoDB i baze na SQL shemu poput MySQL su s vremenom obe postale hibridne baze podataka. Zaključak ovog rada je da svaka baza podataka i shema pohranjivanja ima svoje prednosti i nedostatke te je potrebno proučiti prirodu podataka koje pohranjujemo i napravimo bazu sukladnu tome.

Literatura

Open Source Initiative. „Open Source Licenses“. URL: www.opensource.org/licenses [05.09.2018.]

Kemić, Želimir. (2013) *Primjer baze podataka u sustavu MySQL*. Završni rad. Varaždin: Sveučilište u Zagrebu , Fakultet organizacije i informatike

Peroković, Marko. (2014) *Primjena NoSQL baza podataka na sustav za upravljanje dokumentima*. Diplomski rad. Varaždin: Sveučilište u Zagrebu, Fakultet organizacije i informatike

Vranić, Saša. (2009) *Sučelje katastarske baze podataka.* Diplomski rad. Zagreb: Sveučilište u Zagrebu, Geodetski fakultet

Dadić, Tonči. (2012) *Baze podataka.* Završni rad. Split: Sveučilište u Splitu, Fakultet prirodoslovno- matematičkih znanosti

Welcome to Python. URL: https://www.python.org/

[05.09.2018.]

Willer, Mirna. UNIMARC u teoriji i praksi. Rijeka : Naklada Benja, 1996. Poglavlje 1 Formati za strojno čitljivo katalogiziranje i 2.1-2.6 UNIMARC i standardizacija.

Verona, E. Međunarodni ured za univerzalnu bibliografsku kontrolu (International Office for UBC). // Vjesnik bibliotekara Hrvatske 20(1974)1-4, 50-52.

Willer, M. UNIMARC u teoriji i praksi. Rijeka : Benja, 1996. Poglavlje: 2.3 Standardizacija bibliografskog opisa, str. 61-64.

Verona, E. Međunarodni ured za univerzalnu bibliografsku kontrolu (International Office for UBC). // Vjesnik bibliotekara Hrvatske 20(1974)1-4, 50-52.

ISBD International Standard Bibliographic Description. IFLA series on bibliographic control. München: K.G. Saur. 2011

Willer, Mirna. UNIMARC u teoriji i praksi. Rijeka : Naklada Benja, 1996. Poglavlje 1 Formati za strojno čitljivo katalogiziranje i 2.1-2.6 UNIMARC i standardizacija.

Willer, Mirna. UNIMARC Skraćeni bibliografski format. Zagreb: Hrvatsko knjižničarsko društvo, 2009.

Denton, W. FRBR and the history of cataloging. // Understanding FRBR: what it is and how it will affect our retrieval tools / edited by A. G. Taylor. Westport, CT : Libraries Unlimited, 2007. Str. 35-57.

Dodatak

Kod za dobavu:

```
import urllib.request
import json

# komponente za poziv na API
url_pattern                                                                    =
'https://openlibrary.org/api/books?bibkeys={}&jscmd=details&format=json'
prefix = "ISBN:"

# ISBN-ovi za koje će se skinuti podaci
isbns = [
    '9780515134483', '9780451190529', '9780312363055', '9780451210869',
    '9780385504201', '9780765356130', '9780345339737', '9781439133941',
    '9780786838653', '9780545060394', '9780345348104', '9781570086472',
    '9781416901945', '9780440227533', '9780439372978', '9780385732550',
    '9780553582017', '9780385490818', '9781593080273', '9780756404734',
    '9780688120498', '9780441005901'
]

# stvori URL za dohvat podataka preko ISBN-a
parts = []
for isbn in isbns:
    parts.append(prefix + isbn)
url = url_pattern.format(','.join(parts))

# dohvati podatke i sačuvaj ih kao book_data.json
response = urllib.request.urlopen(url)
content = response.read()
content = content.decode('utf-8')
response.close()
with open('./book_data.json', 'w', encoding='utf-8') as file:
    file.write(content)
```

SQLite baza kod:

```
import sqlite3
```

```
# SQL DIO
table_names = ['BOOKS', 'AUTHORS', 'PUBLISHERS', 'BOOKAUTHOR']
drop_pattern = 'DROP TABLE IF EXISTS {};'

create_books = """
CREATE TABLE BOOKS (
    Book_id INTEGER PRIMARY KEY,
    Title TEXT NOT NULL,
    Subtitle TEXT,
    Series TEXT,
    ISBN10 TEXT,
    ISBN13 TEXT,
    Pages TEXT,
    Publish_date,
    Publisher_id INTEGER
);
"""

create_authors = """
CREATE TABLE AUTHORS (
    Author_id  INTEGER PRIMARY KEY,
    Author TEXT NOT NULL
);
"""

create_bookauthors = """
CREATE TABLE BOOKAUTHOR (
    Book_id INTEGER,
    Author_id INTEGER,
    PRIMARY KEY(Book_id, Author_id)
);
"""

create_publishers = """
CREATE TABLE PUBLISHERS (
    Publisher_id INTEGER PRIMARY KEY,
    Publisher TEXT NOT NULL,
    City TEXT
);
"""

create_tables = [create_books, create_publishers,
                 create_authors, create_bookauthors]
```

```
# STVORI BAZU I IZVRŠI SQL
sqlite_file = './BiblioBaza.sqlite'

conn = sqlite3.connect(sqlite_file)
c = conn.cursor()

for name in table_names:
    c.execute(drop_pattern.format(name))

for create in create_tables:
    c.execute(create)

conn.commit()
```

MongoDB kod:

```
import pymongo
import json

# usnimi podatke
data_path = './book_data.json'
with open(data_path, encoding='utf-8') as file:
    data = json.load(file)

myclient = pymongo.MongoClient("mongodb://localhost:27017/Project")
mydb = myclient["Baza"]
mycol = mydb["Books"]

book_count = 0
for key in data:
    book_data = data[key]['details']
    book_count += 1
    book_id = book_count
    title = book_data['title']
    subtitle = book_data.get('subtitle')
    pages = book_data.get('number_of_pages')
    publish_date = book_data.get('publish_date')
    series = book_data.get('series')
    if series:
        series = ','.join(series).replace(';', '')
    isbn_10 = book_data.get('isbn_10')
    if isbn_10:
        isbn_10 = ','.join(isbn_10)
```

```
    isbn_13 = book_data.get('isbn_13')
    if isbn_13:
        isbn_13 = ','.join(isbn_13)

    publisher = book_data.get('publishers')
    if publisher:
        publisher = ';'.join(publisher)

    authors = data.get('authors')
    author_value = []
    if authors:
        for author_data in authors:
            author_value.append(author_data['name'])

    book_document = {
        "_id": book_id, "Title": title, "Subtitle": subtitle,
        "Series": series, "Author": author_value,
        "Publisher": publisher, "Publish date": publish_date,
        "ISBN10": isbn_10, "ISBN13": isbn_13,
        "Pages": pages
    }

    x = mycol.insert_one(book_document)
```

SQLite kod:

```
import json
import sqlite3

# usnimi podatke
data_path = './book_data.json'
with open(data_path, encoding='utf-8') as file:
    data = json.load(file)

# spoji se na bazu
sqlite_file = 'BiblioBaza.sqlite'
conn = sqlite3.connect(sqlite_file)
c = conn.cursor()

# definiraj potreban SQL
sql_aut = "INSERT INTO AUTHORS ('Author_id','Author') VALUES ({},'{}');"
sql_book_aut = "INSERT INTO BOOKAUTHOR ('Book_id','Author_id') VALUES ({},{});"
```

```
sql_pub = "INSERT INTO PUBLISHERS ('Publisher_id','Publisher','City') VALUES
({},'{}','{}');"
sql_book = """INSERT INTO BOOKS
('Book_id','Title','Subtitle','Series','ISBN10','ISBN13',
'Pages','Publish_date','Publisher_id')
VALUES ({},'{}','{}','{}','{}','{}','{}','{}',{});"""

# šifre za knjige, autore i izdavače
book_count = 0
author_count = 0
publisher_count = 0

# pamti već viđene izdavače i autore
author_registry = {}
publisher_registry = {}

for key in data:
    book_data = data[key]['details']
    book_count += 1
    book_id = book_count
    title = book_data['title']
    subtitle = book_data.get('subtitle')
    pages = book_data.get('number_of_pages')
    publish_date = book_data.get('publish_date')
    series = book_data.get('series')
    if series:
        series = ','.join(series).replace(';', '')
    isbn_10 = book_data.get('isbn_10')
    if isbn_10:
        isbn_10 = ','.join(isbn_10)
    isbn_13 = book_data.get('isbn_13')
    if isbn_13:
        isbn_13 = ','.join(isbn_13)

    publisher = book_data.get('publishers')
    if publisher:
        publisher = ';'.join(publisher)
        if publisher in publisher_registry:
            publisher_id = publisher_registry[publisher]
        else:
            # create new publisher
            publisher_count += 1
            publisher_id = publisher_count
            publisher_registry[publisher] = publisher_id
```

```
            publisher_city = book_data.get('publish_places')
            if publisher_city:
                publisher_city = ';'.join(publisher_city)
            else:
                publisher_city = None
            c.execute(sql_pub.format(publisher_id, publisher, publisher_city))
    else:
        publisher_id = None

    authors = data.get('authors')
    if authors:
        for author_data in authors:
            name = author_data['name']
            if name in author_registry:
                author_id = author_registry[name]
            else:
                author_count += 1
                author_id = author_count
                author_registry[name] = author_id
                # add a new author
                c.execute(sql_aut.format(author_id, name))
            # add a relation between an author and a book
            c.execute(sql_book_aut.format(book_id, author_id))

    # add book data
    bk_sql = sql_book.format(book_id, title, subtitle, series, isbn_10,
                             isbn_13, pages, publish_date, publisher_id)
    c.execute(bk_sql)
```

Summary

Hierarchical and relational models as bibliographical databases

This work questions the nature of bibliographic data and explores their posibilities of storage in different dana structures. Relational and hierarchy structures of data both have their flaws and advantages. Advantages of storing dana in relational database come forth during storing large amount of dana which are often changed and updated, it allows change of single instance of data which is then updated through all connected data. Main difference of relational databases is the exclusion of repeating dana, which also gives relational databases an edge when it comes to memory usage. This structure makes relational databases very popular in the areas which handle large about of data changing on a regular basis. Advantages

of hierachical database is the possibility of connecting all of our data to a singe object, which makes such a database much easier to use. Every object in the hierarchical database in an object in intself, while in relational database its necessary to call for data from from a number of different tables. The big flaw of this data structure is the number of repeating data is impossible to regulate. Surely the number of repeating data depents on the type of bibliographical data which we're putting in the database, if the subject is of belletristic nature, the number of authors per work is probably gonna stay around one work each, sometimes maybe more. If the subject is scientific works, where one works has up to 30-40 authors, which are repeating in the other works aswell, then we have a problem. Despite all that I believe that hierarchical structure is more ideal for bibliographic work. It describes one work in its entirety and allows of creation of new sub-categories for more specific works. Ideal solution would warrant the implementation of relational structure in areas of often used repeating data, like publisher or author. Luckly most of the databases in use today are hybrid databases, allowing just that, connecting multiple structures of data. Dataases which are imagined as NoSQL databases like MongoDB and databases using SQL shema like MySQL, overtime became hybrid databases. The conclusion of this work is that each database and shema has its own flaws and advantages, and its necessary to study the nature of data we're storing and construct a database according to data.

Key words: Hierarchical models, relational models, bibliography, databases, ISBD, MARC, UNIMARC, SQL, NoSQL, MongoDB, JSON, BSON, FRBR