



Prolećni semestar 2019/2020

Sistem za biblioteke

IT350
Baze podataka

Projektna dokumentacija

Student:

Nikola Tasić 3698

Asistent:

Veljko Grković

Sadržaj

| | |
|------------------------------|----|
| Apstrakt..... | 3 |
| Opis problema..... | 3 |
| Zahtevani upiti | 3 |
| Uvod..... | 4 |
| Cilj..... | 4 |
| Realizacija..... | 4 |
| Projektno rešenje..... | 5 |
| Konceptualni model | 5 |
| Fizički model..... | 6 |
| Kreiranje baze | 7 |
| Generisanje C koda | 8 |
| Parsiranje DDL fajlova | 8 |
| C Header fajl..... | 9 |
| C Source fajl | 10 |

Apstrakt

Opis problema

Opis problema: Državna uprava je odlučila da im je potreba aplikacija za potrebe svih državnih biblioteka. Aplikacija mora da čuva podatke o svim bibliotekama po opštinama u kojima se nalaze. Za svaku opštinu može se zaključiti kom regionu pripada i dobiti naziv regiona. Naziv opštine kojoj biblioteka pripada predstavlja deo kompletne adrese biblioteke. Svaka adresa ima i broj zgrade i ime ulice u kojoj se biblioteka nalazi. Postoji evidencija o zaposlenim u bibliotekama, svaki zaposleni ima ime, prezime, poziciju i biblioteku u kojoj radi. Moguće pozicije su bibliotekar, čistač i direktor. Postoji evidencija o knjigama. Svaka knjiga ima autore, ime, godinu izdanja i ISBN. Svaki autor ima ime, prezime i opis. Jedna knjiga može da ima više autora. Takođe jedan naslov knjige može da ima više primeraka. Postoje korisnici biblioteke. Svaki korisnik ima korisničko ime i šifru. Jedna knjiga može da bude iznajmljena od strane jednog korisnika ali svaki korisnik može da ima i do dve iznajmljene knjige. Potrebno je voditi evidenciju koji korisnik je iznajmio koju knjigu.

Zahtevani upiti

- a. Broj zaposlenih, kao i broj knjiga po biblioteci i gradu
- b. Autore koji imaju preko 10 objavljenih knjiga od 2000. godine do trenutnog datuma
- c. Adrese svih biblioteka gde se može naći pet ili više knjiga autora "Joshua Bloch" 12 e. Uraditi statistički prikaz, na dnevnom nivou koliko je knjiga u kojoj biblioteci iznajmljeno, a koliko vraćeno.
- d. Za biblioteku u kojoj ima najveći broj različitih naslova (ne knjiga/primeraka) prikazati adresu, grad i ukupan broj zaposlenih. Ako postoje dve ili više biblioteke sa istim, najvećim brojem knjiga prikazati podatke za sve takve biblioteke.
- e. Napisati pogled koji za svaki naslov prikazuje broj primeraka koji se nalaze u biblioteci. Prikazati samo one naslove koji imaju više od jednog autora. Statistiku uraditi samo za biblioteke koje se nalaze u Sremskom okrugu.
- f. Prikazati naslov i autora knjige koji su kao nove pristigle u biblioteku. Prikazane knjige su u biblioteku došle one kalendarske godine kada je prikaz napravljen (upit izvršen). Uraditi prikaz samo za biblioteku koja ima najveći broj korisnika. Knjige sortirati po naslovu i godini izdanja.

Uvod

Cilj

Cilj projekta je da se dizajnira sistem koji pruža uvid u poslovanje biblioteka u jednoj državi i omogućava pregled svih knjiga i njihovih autora zajedno sa informacijama da li su te knjige trenutno izdate na čitanje nekom od korisnika.

Realizacija

Baza podataka će biti napravljena za **MariaDB** koja je trenutna open-source implementacija **MySQL**-a za Linux i Windows sisteme. Dizajniranje baze će biti odrađeno preko **Powerdesigner**-a.

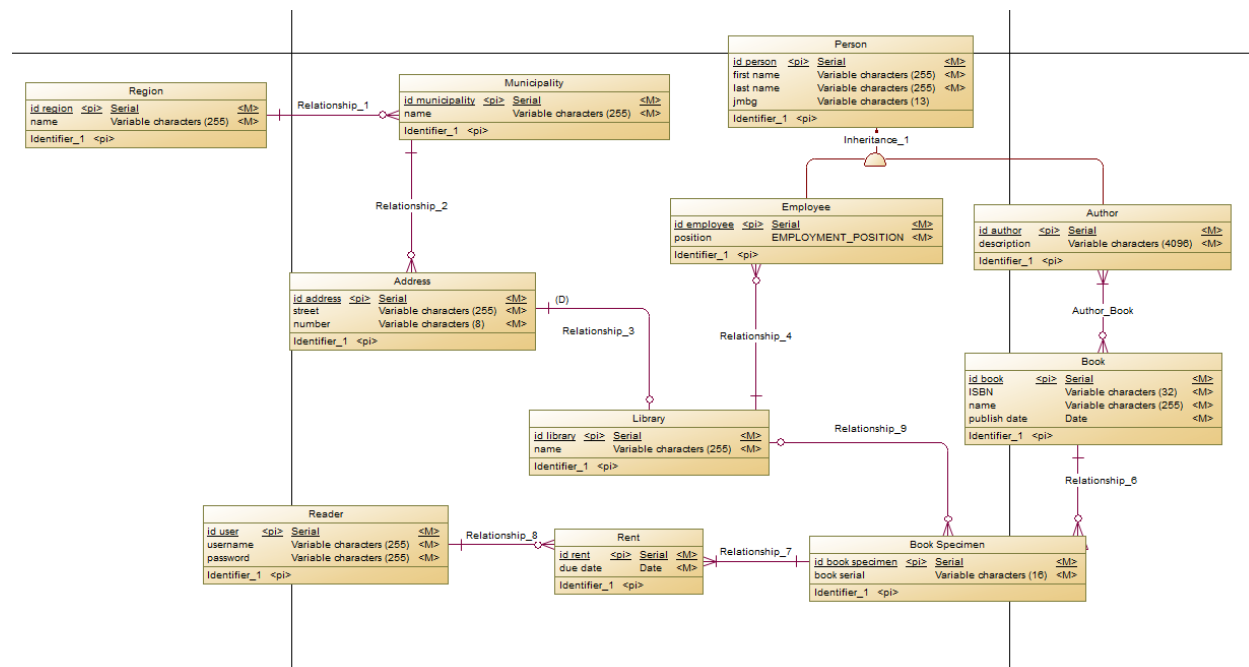
CRUD aplikacija će biti **CLI** (command-line) aplikacija izrađena u programskom jeziku C samo iz pomoć odgovarajuće biblioteke koja je u ovom slučaju **libmysql**. Lični dodatak na ovu problematiku je da će sav kod koji obradjuje logiku CRUD operacija nad bazom biti generisan automatski na osnovu DDL fajla koji predstavlja strukturu baze. Generisanje C koda će izvršiti **Python** program koji je zaduzen za struktuiranje **DDL** opisa u odgovarajuće C header i source fajlove. Među sloj između Python aplikacije i DDL-a je **NodeJS** skripta koji dati DDL čisti od komentara i parsira u **JSON** fajl koji kasnije obrađujemo. Za dizajn CLI interfejsa koristimo dobro poznatu C biblioteku **libncurses**.

Projektno rešenje

Sledi opis projektnog rešenja od šeme baze do implementacije i primera korišćenja aplikacije.

Konceptualni model

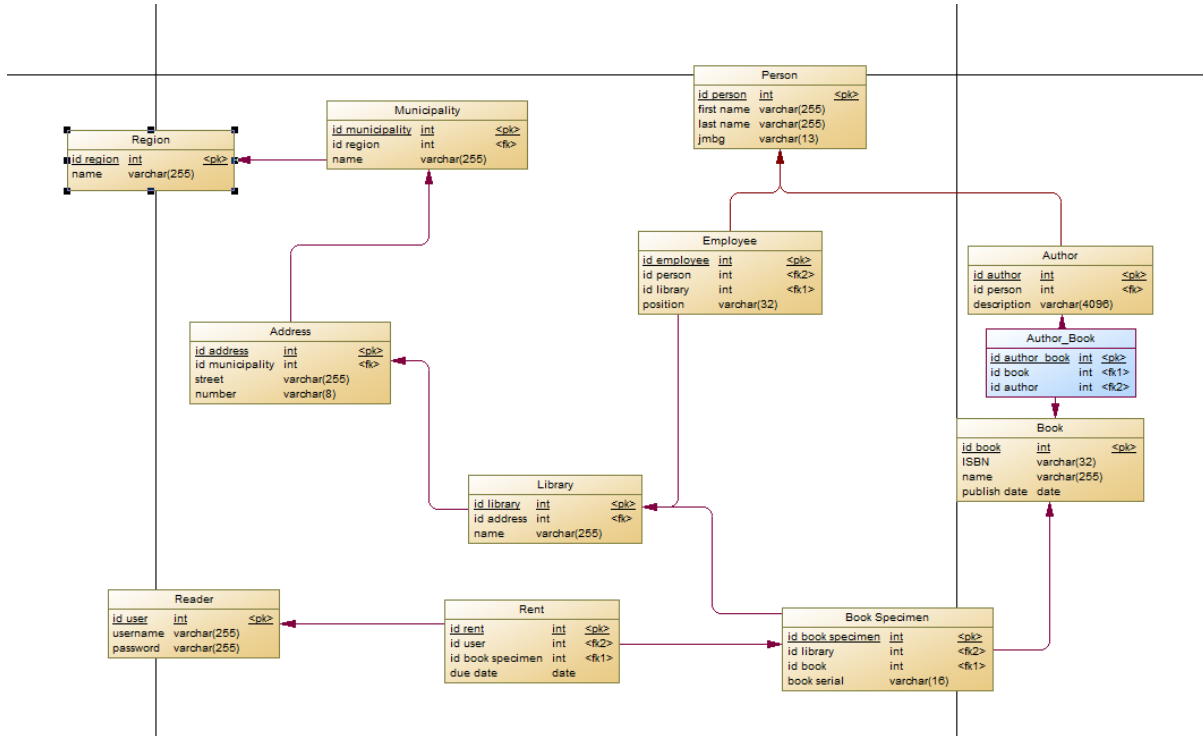
Konceptualni model predstavlja prvi nivo apstrakcije problema dobijenog iz apstrakta.



Slika 1 Konceptualni model baze podataka

Fizički model

Na osnovu konceptualnog modela možemo da generišemo fizički model koji je jedan korak bliži generisanju SQL DDL koda koji ćemo koristiti za kreiranje baze.



Slika 2 Fizički model baze podataka

Kreiranje baze

Na kraju iz fizičkog modela možemo da generišemo SQL kod koji ćemo da koristimo za kreiranje.

```
/*=====*/
/* Table: address */
/*=====*/
create table address
(
    id_address          int not null auto_increment,
    id_municipality     int not null,
    street              varchar(255) not null,
    number              varchar(8) not null,
    primary key (id_address)
);

/*=====*/
/* Table: author */
/*=====*/
create table author
(
    id_author           int not null auto_increment,
    id_person           int not null,
    description         varchar(4096) not null,
    primary key (id_author)
);

/*=====*/
/* Table: author_book */
/*=====*/
create table author_book
(
    id_author_book      int not null,
    id_book             int not null,
    id_author           int not null,
    primary key (id_author_book)
);
```

Slika 3 Primer generisanog DDL koda

Iz ovako generisanog SQL fajla mozemo veoma lako kreirati bazu i sve odgovarajuće tabele **source** komandom.

Generisanje C koda

Kao što je već pomenuto sav kog koji predstavlja logiku vršenja upita nad bazom podataka biće dinamički generisan. Ovakav pristup simulira neku vrstu ORM frejmworka a i takođe olaksava sam proces jer je kod potreban za izvršavanje upita veoma repetativan.

Parsiranje DDL fajlova

Na osnovu DDL-a i libmysql dokumentacije mozemo prilično lako da definemo konverziju tipova ova dva veoma različita sistema.

```
class SqlType(enum.Enum):
    LONG = 1,
    VARCHAR = 2,
    FK_LONG = 3,
    PK_LONG = 4,
    DATE = 5,
    TEXT = 6,

    def tostring(self):
        if self == SqlType.PK_LONG:
            return "MYSQL_TYPE_LONG"
        if self == SqlType.FK_LONG:
            return "MYSQL_TYPE_LONG"
        if self == SqlType.LONG:
            return "MYSQL_TYPE_LONG"
        if self == SqlType.VARCHAR:
            return "MYSQL_TYPE_STRING"
        if self == SqlType.DATE:
            return "MYSQL_TYPE_DATE"
        if self == SqlType.TEXT:
            return "MYSQL_TYPE_STRING"
```

Slika 4 Enum reprezentacija DDL tipova

```
if {"column": col_name} in table["primaryKey"]["columns"]:
    col_type_sql = SqlType.PK_LONG
elif col_type == "varchar":
    col_type_sql = SqlType.VARCHAR
elif col_type == "char":
    col_type_sql = SqlType.VARCHAR
elif col_type == "text":
    col_type_sql = SqlType.TEXT
elif col_type == "int":
    col_type_sql = SqlType.LONG
elif col_type == "date":
    col_type_sql = SqlType.DATE
else:
    msg = f"SQL type not handled '{col_type}' in struct '{struct_name}'"
    assert False, msg

if "foreignKeys" in table.keys():
    if {"column": col_name} in [fk["columns"][0] for fk in table["foreignKeys"]]:
        col_type_sql = SqlType.FK_LONG
elif "indexes" in table.keys():
    if {"column": col_name} in [fk["columns"][0] for fk in table["indexes"]]:
        col_type_sql = SqlType.FK_LONG
```

Slika 5 Parsiranje JSON-a u enum

U slici broj 5 vidimo da naravno nisu svi tipovi podržani ali možemo lako da dodajemo nove. U tome nam pomaze **assert** koji se okida ako naiđemo na neki tip koji nije podržan. Kada se kod izvrši bez ijednog asserta onda znači da smo implementirali svaku potrebnu metodu. Ovakav pristup je verovatno lakše implementirati preko neke vrste interfejsa ali rezultati parsiranja JSON-a i kod koji je potreban za izvršavanje svih upita ne može baš tako lako da se generalizuje. Ovakav pristup pročešljavanja grešaka je bio najlakši za implementaciju.

C Header fajl

Iz enuma je lako generisati C header fajl.

```
class Header:
    def __init__(self):
        self.global_includes: List[str] = []
        self.local_includes: List[str] = []
        self.pragmas: List[str] = []
        self.macros_def: List[MacroDefinition] = []
        self.functions: List[FunctionTemplate] = []
        self.structs = []
        self.header_guard = None
```

Slika 6 Python reprezentacija C header fajla

Ovakva struktura nam omogućuje da dinamički dodajemo sve makroe koji su nam potrebni da tipa definisemo header guardove i importove (#include) ostalih header fajlova koji su nam potrebni za komunikaciju. Takođe suštinski deo svakog header fajla su funkcije koje isto sa lakoćom možemo dodati preko klase **FunctionTemplate**.

Rezultat jedne ovakve klase je na primier ovakav C header fajl. Svaka od FK veza iz DDL-a je predstavljena kao pointer do nove strukture dok su ostala polja parsirana u odgovarajuće C tipove. Obraćamo pažnju na to da svako tekstualno polje ima +1 veću dužinu od definisanog. To je potrebno da bi smo uključili u obzir '0' (null) karakter za terminaciju stringa.

```
#include "db/orm/entity.h"

struct employee {
    uint id_employee;
    struct person* person;
    struct library* library;
    char position[33];
};
```

C Source fajl

```
/* Generated function */
uint municipality_insert(MYSQL* conn, MUNICIPALITY* municipalityT) {
    #define QUERY_LENGTH 512
    #define STRING_SIZE 255
    #define QUERY "insert into municipality (id_region, name) values (?, ?);"
    #define PARAM_COUNT 2
    #define NAME_SIZE 256
    /* Generated using get_insert_assertions() */
    assert(municipalityT->id_municipality == 0);
    assert(strlen(municipalityT->name, STRING_SIZE) > 1);
```

Slika 7 Primer C source fajla i metode za INSERT

Naravno u header fajl dolazi C izvorni fajl koji sadrži sve osnovne metode za CRUD operacije.

```
/* Generated using get_col_param_buffers() */

/* INTEGER PARAM */
param[0].buffer = malloc(sizeof(uint));
param[0].buffer_type = MYSQL_TYPE_LONG;
memcpy(param[0].buffer, &municipalityT->region->id_region, sizeof(uint));
/* STRING PARAM */
param[1].buffer = malloc(name_len);
param[1].buffer_type = MYSQL_TYPE_STRING;
param[1].buffer_length = name_len;
strncpy(param[1].buffer, municipalityT->name, name_len);
```

Slika 8 Primer pripreme buffera koji izvršava upit

```
retval = (uint) mysql_stmt_insert_id(stmt);

// update id after insertion;
municipalityT->id_municipality = retval;

/* Generated using col_param_buffer_free() */
free(param[0].buffer);
free(param[1].buffer);

return retval;
```

Slika 9 Oslobađanje memorije buffera i azuriranje ID-a

Rešenja zahtevanih upita

Upit a

Broj zaposlenih, kao i broj knjiga po biblioteci i gradu

```
select library.name, municipality.name as 'Municipality', count(book_specimen.id_book_specimen) as 'Books'
from book_specimen
    join library on library.id_library = book_specimen.id_library
    join address on address.id_address = library.id_address
    join municipality on address.id_municipality = municipality.id_municipality
group by library.id_library;
```

```
select library.name, municipality.name as 'Municipality', count(employee.id_employee) as 'Employees'
from employee
    join library on library.id_library = employee.id_library
    join address on address.id_address = library.id_address
    join municipality on address.id_municipality = municipality.id_municipality
group by library.id_library;
```

Upit b

Autore koji imaju preko 10 objavljenih knjiga od 2000. godine do trenutnog datuma

```
select book_authors.id_author, person.first_name, person.last_name, `Books Published`
from (select author.id_author, author.id_person, count(books.id_book) as `Books Published`
    from (select id_author, author_book.id_book, publish_date
        from author_book
            join book b on author_book.id_book = b.id_book
        where (b.publish_date between cast(2000-01-01' as date) and curdate())) as `books`
    join author on books.id_author = author.id_author
group by author.id_author) as book_authors
    join person on person.id_person = book_authors.id_person
where `Books Published` > 10;
```

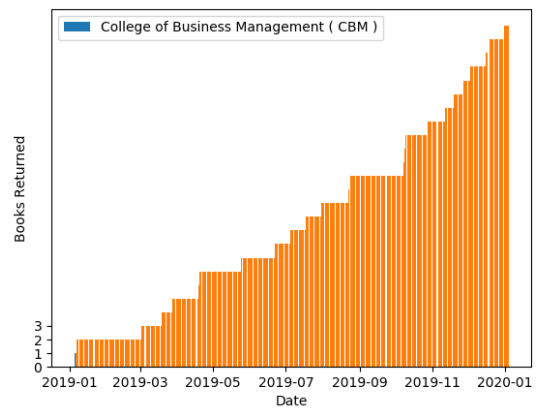
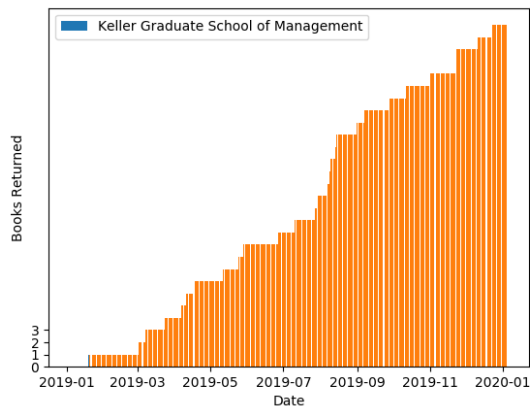
Upit c

Adrese svih biblioteka gde se može naći pet ili više knjiga autora "Joshua Bloch"

```
select `Count`, library.name, street, number
from (select book_specimen.id_book_specimen,
            book_specimen.id_library,
            one_author.id_book,
            one_author.id_author,
            count(book_specimen.id_book_specimen) as `Count`
from (select book.id_book, ab.id_author
      from book
        join author_book ab on book.id_book = ab.id_book
        join author a on ab.id_author = a.id_author) as one_a
      join book_specimen on one_author.id_book = book_specimen.id
      join library on one_author.id_library = library.id_library
      join address a2 on library.id_address = a2.id_address
      join person on person.id_person = author.id_person
      group by one_author.id_book) as libraries
where person.first_name = 'Joshua'
      and person.last_name = 'Bloch'
      and `Count` >= 5;
```

Upit d

Uraditi statistički prikaz, na dnevnom nivou koliko je knjiga u kojoj biblioteci iznajmljeno, a koliko vraćeno.



Obzirom na to da se trazio statisticki prikaz rešenje je odrađeno uz pomoć Python biblioteke **matplotlib**.

Upit e

Za biblioteku u kojoj ima najveći broj različitih naslova (ne knjiga/primeraka) prikazati adresu, grad i ukupan broj zaposlenih. Ako postoje dve ili više biblioteke sa istim, najvećim brojem knjiga prikazati podatke za sve takve biblioteke.

```
select top_libraries.id_library,
       name,
       street,
       number,
       count(top_libraries.id_library) as `No. Employees`,
       `Title Count`
from (select id_library, name, street, number, max(`Title Count`) as `Title Count`
      from (select *, count(id_library) as `Title Count`
            from (select library.id_library, library.name
                  from library
                  join book_specimen bs on library.id_library = bs.id_library
                  join book b on bs.id_book = b.id_book
                  group by isbn, library.id_library
                  order by library.id_library) as unique_books
            group by name
            order by `Title Count` desc) as libraries
      join address on address.id_address = libraries.id_library)
as top_libraries
join employee on top_libraries.id_library = employee.id_library
group by employee.id_library;
```

Upit f

Napisati pogled koji za svaki naslov prikazuje broj primeraka koji se nalaze u biblioteci. Prikazati samo one naslove koji imaju više od jednog autora. Statistiku uraditi samo za biblioteke koje se nalaze u Sremskom okrugu.

```
create view `Books Per Library` as
select id_library, books3.name as `Book Name`, isbn, `Library Name`, id_book, `Specimen Count`, id_address
from (select *, count(isbn) as `Specimen Count`
      from (select *
            from (select library.id_library,
                        library.id_address,
                        library.name as `Library Name`,
                        id_book_specimen,
                        bs.id_book as idbook
                   from library
                   join book_specimen bs on library.id_library = bs.id_library
                   join book b on bs.id_book = b.id_book) as books2
            join book b on b.id_book = books2.idbook
            group by isbn) as books3;

select `Book Name`, isbn, `Library Name`, `Specimen Count`, `Authors`, m.name as `Municipality Name`
from (select id_library, `Book Name`, isbn, `Library Name`, `Specimen Count`, count(id_author) as `Authors`, id_address
      from `Books Per Library`
      join author_book ab on `Books Per Library`.id_book = ab.id_book
      group by `Book Name`) as `Books and Authors`
join address on address.id_address = `Books and Authors`.id_address
join municipality m on address.id_municipality = m.id_municipality
where `Authors` > 1
and m.name = 'Sremski okrug';
```

Upit g

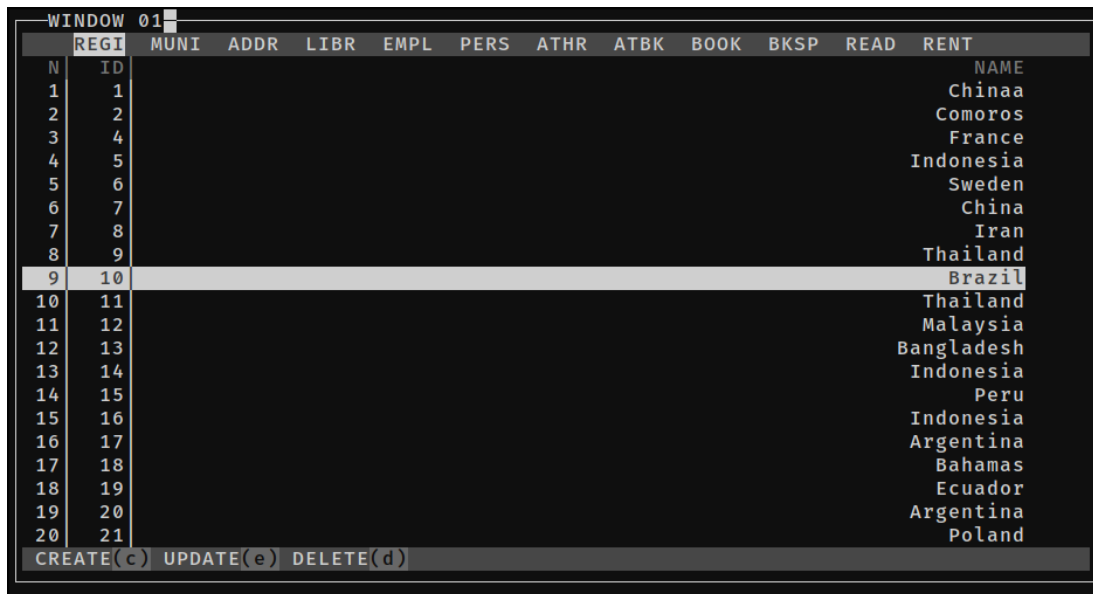
Prikazati naslov i autora knjige koji su kao nove pristigle u biblioteku. Prikazane knjige su u biblioteku došle one kalendarske godine kada je prikaz napravljen (upit izvršen). Uraditi prikaz samo za biblioteku koja ima najveći broj korisnika. Knjige sortirati po naslovu i godini izdanja.

```
select id_library, `Library Name`, `Readers`
from (select reader.id_reader, library.id_library, library.name as `Library Name`, count(reader.id_reader) as 'Readers'
      from reader
           join rent on reader.id_reader = rent.id_reader
           join book_specimen bs on bs.id_book_specimen = rent.id_book_specimen
           join library on library.id_library = bs.id_library
      group by library.id_library) as `Readers per Library`
group by id_library order by `Readers` desc;
```


Aplikacija

U ovom delu cemo napraviti osnovni pregled izgelda i funkcionalnosti CRUD aplikacije. Aplikaciju pokrecemo iz komandne linije pomocu `./it350_pz_app`

List view (READ)



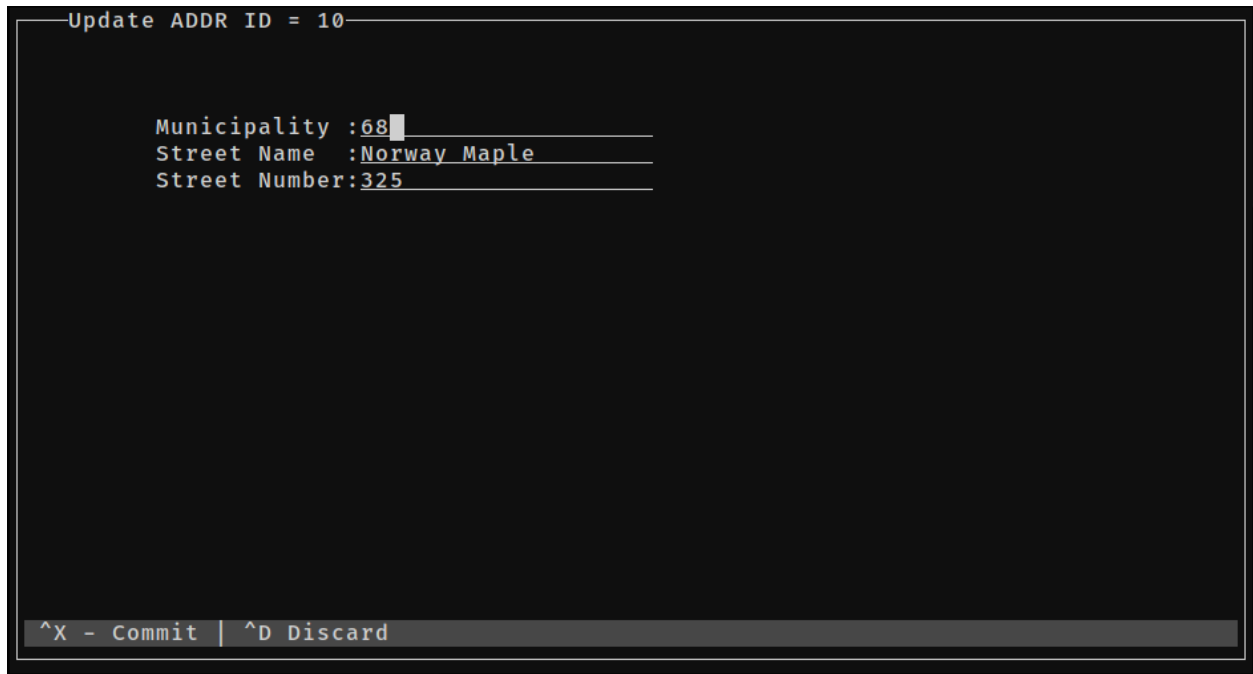
| N | REGI | MUNI | ADDR | LIBR | EMPL | PERS | ATHR | ATBK | BOOK | BKSP | READ | RENT | NAME |
|----|------|------|------|------|------|------|------|------|------|------|------|------|------------|
| 1 | 1 | | | | | | | | | | | | Chinaa |
| 2 | 2 | | | | | | | | | | | | Comoros |
| 3 | 4 | | | | | | | | | | | | France |
| 4 | 5 | | | | | | | | | | | | Indonesia |
| 5 | 6 | | | | | | | | | | | | Sweden |
| 6 | 7 | | | | | | | | | | | | China |
| 7 | 8 | | | | | | | | | | | | Iran |
| 8 | 9 | | | | | | | | | | | | Thailand |
| 9 | 10 | | | | | | | | | | | | Brazil |
| 10 | 11 | | | | | | | | | | | | Thailand |
| 11 | 12 | | | | | | | | | | | | Malaysia |
| 12 | 13 | | | | | | | | | | | | Bangladesh |
| 13 | 14 | | | | | | | | | | | | Indonesia |
| 14 | 15 | | | | | | | | | | | | Peru |
| 15 | 16 | | | | | | | | | | | | Indonesia |
| 16 | 17 | | | | | | | | | | | | Argentina |
| 17 | 18 | | | | | | | | | | | | Bahamas |
| 18 | 19 | | | | | | | | | | | | Ecuador |
| 19 | 20 | | | | | | | | | | | | Argentina |
| 20 | 21 | | | | | | | | | | | | Poland |

CREATE(c) UPDATE(e) DELETE(d)

Slika 10 Izgled osnovnog list view-a

U ovom view-u imamo pregled svih tabela iz baze do kojih mozemo da navigiramo pomoću strelica levo i desno dok selekciju vršimo strelicama gore i dole. Na dnu prozora se nalazi informacije koje komande korsitimo za CRUD operacije. Pritiskom na dugme 'L' otvaramo novi identični List View koji može da nam posluži ako zelimo da napravimo brzu promenu u nekoj drugoj tabeli.

Form view (UPDATE)



```
Update ADDR ID = 10

Municipality :68
Street Name  :Norway Maple
Street Number:325

^X - Commit | ^D Discard
```

Slika 11 Primer UPDATE interfejsa

Posle popunjavanja podataka koristimo CTRL+X da sačuvamo podatke (po uzoru na *nano* editor) ili CTRL+D (po uzoru na logout prečicu u linux terminalima) da bi se vratili na početni list view.

Popup view (DELETE)

WINDOW 01

| REGI | MUNI | ADDR | LIBR | EMPL | PERS | ATHR | ATBK | BOOK | BKSP | READ | RENT |
|------|------|------------|------|-----------|------|------|------|------|------|---------------|------|
| N | ID | FIRST NAME | | LAST NAME | | | | | | JMBG | |
| 1 | 1 | Clayborn | | Pengelley | | | | | | 7791537270864 | |
| 2 | 2 | Elita | | Benedict | | | | | | 2436770059354 | |
| 3 | 3 | Issi | | Petrik | | | | | | 4004190050249 | |
| 4 | 4 | Carl | | | | | | | | 4294282547878 | |
| 5 | 5 | Len | | | | | | | | 6432784097173 | |
| 6 | 6 | Ale | | | | | | | | 6771164183617 | |
| 7 | 7 | Mort | | | | | | | | 0964082910404 | |
| 8 | 8 | Lu | | | | | | | | 5288039688538 | |
| 9 | 9 | Beatri | | | | | | | | 4666553786226 | |
| 10 | 10 | Erne | | | | | | | | 6216697415689 | |
| 11 | 11 | Jac | | | | | | | | 5015471286482 | |
| 12 | 12 | Cynd | | | | | | | | 2022692786501 | |
| 13 | 13 | Rah | | | | | | | | 7659878016575 | |
| 14 | 14 | Ile | | | | | | | | 2315822188029 | |
| 15 | 15 | Kateri | | | | | | | | 4288011509918 | |
| 16 | 16 | Jordain | | Longmead | | | | | | 2094205435954 | |
| 17 | 17 | Marguerite | | Sawforde | | | | | | 3551244288174 | |
| 18 | 18 | Idaline | | Hatzar | | | | | | 0212921197800 | |
| 19 | 19 | Meade | | Mc Curlye | | | | | | 5614612921342 | |
| 20 | 20 | Reinold | | Rate | | | | | | 8961290755966 | |

WINDOW 02

Are you sure you want to delete?

CREATE(c) UPDATE(e) DELETE(d)

Slika 12 Primer popup-a za brisanje podatka

Pritiskom na Y ili N odgovaramo na prosto pitanje.

Form view (CREATE)

Add a new READ

Username :username

Password :password123

^X - Commit | ^D Discard

Slika 13 Sličan primer kao kod UPDATE forme

Zaključak

Projekat realizuje osnovne CRUD operacije nad definisanom bazom podataka. Akcenat je naravno bačen na automatsko generisanje koda i parsiranje DDL-a što uvodi malo kompleksnosti u inače trivijalan i monoton posao dizajniranja CRUD aplikacije. Dizajn baze je bio takav da se što je više moguće smanje redundantnost podataka što je i postignuto. Sama aplikacija naravno je veoma bazična ali daje dobar proof-of-concept za gore pomenutu automatsko generisanje i parsiranje.

Bibliografija

Duarte, A. F. (n.d.). *SQL DDL to JSON Schema*. Retrieved from npm:
<https://www.npmjs.com/package/sql-ddl-to-json-schema>

Guittet, T. (n.d.). *fields_magic.c*. Retrieved from GitHub: <https://gist.github.com/alan-mushi/c8a6f34d1df18574f643>

Hall, J. (n.d.). *Getting started ncurses*. Retrieved from LinuxJournal:
<https://www.linuxjournal.com/content/getting-started-ncurses>

MYSQL C API Reference. (n.d.). Retrieved from MYSQL: <https://dev.mysql.com/doc/refman/8.0/en/c-api-function-overview.html>

Padala, P. (n.d.). *NCURSES programming HOWTO*. Retrieved from tldp:
<http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/>