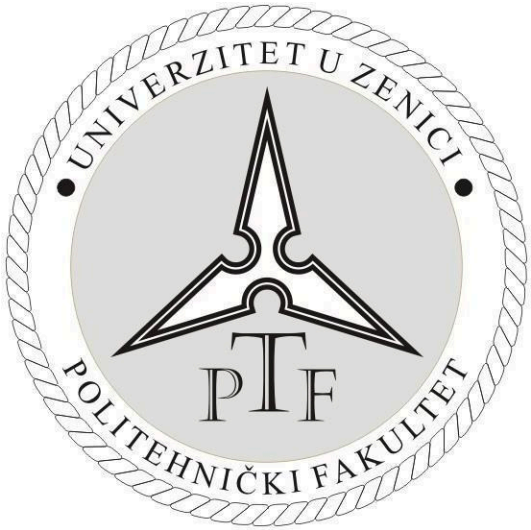

- ZAVRŠNI PROJEKAT –

OBJEKTNO-ORIJENTISANO PROGRAMIRANJE

DOKUMENTACIJA
DATABASE MANAGEMENT SYSTEM

Studenti: **Almir Mujanović (343), Inas Malkić (349),**

Edin Omanović (365), Emma Tica (350)

Profesor: **Adnan Dželihodžić**

Asistenti: **Edin Tabak i Narcisa Hadžajlić**

Akademska godina: 2023 - 2024

1. Osnovni opis projekta:

Naš projekat Sistema za upravljanje bazama podataka (DBMS) je sveobuhvatno rješenje dizajnirano da olakša efikasnu i strukturiranu interakciju sa bazama podataka koristeći prilagođeni jezik upita.

Ono predstavlja jedan pojednostavljeni oblik upitnog jezika SQL i omogućava korisnicima da kreiraju i upravljaju svojim bazama podataka koristeći mnogima već poznatu klasičnu sintaksu SQL-a. Upravljanje tim bazama je omogućeno komandama za kreiranje i brisanje baza, ali i tabela u njima, kao i uređivanje i promjene izgleda tabela i informacija u njima, te naravno unošenje i brisanje samih podataka.

Sistem je zasnovan na AST (Abstract Syntax Tree) i predstavlja naš pokušaj kreiranja projekta koji bi bio što zanimljiviji moguće, a i dalje u skladu s našim znanjem i mogućnostima. Uzimajući u obzir sve okolnosti, može se reći da je ovo jedan prilično složen i napredan projekat na koji smo ponosni.

2. Funkcionalnosti

1. Klase

- **Menu.h**
  Funkcije showMenu i showHelp služe nam za ispis glavnog menija i pomoći(sintaksa komandi).
- **User.h** je klasa koja nam omogućuje rad sa sistemom korisnika od kojih svako ima svoj folder u kojem će se čuvati sve baze podataka zajedno sa svim njihovim podacima,.
  Također je generisan i users.json file u kojem su sačuvani login podaci o svakom korisniku uz naglasak da su password-i sačuvani u enkriptovanom formatu.
- **Tokenizer.h** sadrži enumeraciju TokenType (navedena ispod), strukturu Token (tip tokena i njegova vrijednosti) i vrlo bitnu klasu Tokenizer. Public dio klase Tokenizer se sastoji od: konstruktora Tokenizer(std::string str) : m_string(str){} koji uzima string kao input koji treba "tokenizirati", destruktora i funkcije tokenize() koja generiše vektor "Token" objekata.
  U *private* dijelu klase također imamo bitne funkcije peak() koja traži prvu sljedeću vrijednost (string) naše SQL komande i consume() koja će primiti trenutnu vrijednosti u komandi te preći na sljedeću.
- **Parser.h** sadrži dvije glavne klase NodeStmt i NodeExpr iz kojih je izveden veliki broj klasa koje nam služe za obradu i pozivanje komandi. Iz klase NodeStmt su izvedene klase poput NodeStmtUseDatabase, NodeStmtCreateTable itd.
  Klasa Parser sadrži konstruktor, parseStmt() (parsira jedan statement i vraća pointer na NodeStmt objekat), parseExpression() (parsira jedan expression i vraća pointer na NodeExpr objekat), parseProgram() (parsira niz naredbi i formira program, a zatim vraća pointer na NodeProgram objekat).
- **Generator.h**
  Funkcija genExpr uzima referencu NodeExpr i pretvara je u originalnu vrijednost tipa &expr.
  Funkcija genStmt uzima referencu NodeStmt& i na osnovu tipa Statementa koji dobije, izvršava operacije. To su operacije poput "CREATE DATABASE", "INSERT INTO TABLE" itd.
  Funkcija void Generate() je ta koja na kraju izvršava samu naredbu.

2. *Enumeracija* TokenType sadrži listu svih mogućih tokena koje Parser može primiti:

```
enum class TokenType{
CREATE,DATABASE,IDENTIFIER,LBRACE,RBRACE,COMMA,DELETE,TABLE,USE,INSERT,INTO,SHOW,
DATABASES,TABLES,VALUES,ALTER,ADD,DROP,COLUMN,STRING,INTEGER,INT_DATA_TYPE,
STRING_DATA_TYPE,SELECT,FROM,WHERE,EQUALS,AND,NOT_EQUAL,USER,IDENTIFIED,BY,ROLE,
LOGOUT,UPDATE,SET};
```

3. Ovaj program se dosta oslanja na funkcionalnost pametnih pokazivača, npr
   Parser.cpp - Using std::make_unique:

```
std::unique_ptr<NodeExpr> Parser::parseExpression() {
    const Token token = consume();
    if (token.type == TokenType::IDENTIFIER) {
        if (token.value.has_value()) {
            return std::make_unique<NodeExprIdentifier>(token.value.value());
        }
```

   korištenje pametnih pokazivača pojednostavljuje upravljanje memorijom i čini kod sigurnijim i lakšim za održavanje.

4. Iznimke:
   primjer iznimke se nalazi u main.cpp fajlu:

```
try {
```

```
            Tokenizer tokenizer(input);
            std::vector<Token> tokens = tokenizer.tokenize();
           //pozivanje raznih funkcija u main.cpp...
        } catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
```

Lahko je zaključiti da se iznimka koristi kako bi se izbjegla bilo kakva greška ili neočekivana vrijednost koja bi se mogla javiti u nekoj od pozvanih funkcija.

5. Primjer virtuelne funkcije imamo u Parser.h, u klasi NodeStmt:

```
class NodeExpr {
public:
   virtual ~NodeExpr() {}
   virtual std::unique_ptr<NodeExpr> clone() = 0;
   virtual std::string toString(){
       return "NodeExpr";
}};
```

Ovo je uobičajeni obrazac za postizanje polimorfizma. Omogućava stvaranje kopije objekta bez poznavanja njegove konkretne vrste pri vrijeme kompilacije.

6. preopterećeni operator << koristi se za ispisivanje ažuriranih vrijednosti tabele u output file.

```
class NodeStmtUpdate : public NodeStmt {
public:
   // logika komande update table ..
   std::ostream& operator<<(std::ostream& os) {
       print();
       return os;
```

7. Nasljeđivanje je jedan od ključnih elemenata strukture AST.Najbolji primjer za to je NodeStmt klasa iz koje je izveden veliki broj drugih klasa koje izvršavaju date SQL komande. Sve ove izvedene klase naravno imaju svoju unikatnu funkcionalnost, ali svaka od njih koristi svoju roditeljsku klasu da bi pristupila nekim osnovnim vrijednostima nižeg nivoa.
8. Rad s datotekama je neophodan element DBMS da bi on bio koristan i funkcionalan. Pomoću nlohmann::json biblioteke svi podaci čuvaju se u json formatu:

- users.json za svakog korisnika čuva id, username, password (enkriptovan) i ulogu.
- data folder sadrži subfolder koji nosi naziv ID-ja za svakog korisnika pojedinačno. U svakom folderu specifičnom za određenog korisnika zapisuju se baze podataka u json formatu.

9. Glavni meni ima sljedeći oblik:

```
1. Login
2. Help [COMMANDS]
3. Exit
Enter your option:...
mlinql> …
```

, a njegova struktura je objašnjena iznad.

3. Dodatno

1. Upotreba RTTI (Run-Time Type Information)

   U Generator.cpp koristimo dynamic_cast za provjeru tipa objekta u runtime-u. Ovo omogućava da se dinamički provjeri tip izraza ili izjave i da se shodno tome izvrši određena logika.

2. Upotreba std::optional iz standardne biblioteke

   std::optional omogućava da funkcija peak() vrati neodređenu vrijednost kada nema više tokena za obraditi. Ovo je korisno u scenarijima gdje je kraj niza tokena očekivan i validan ishod, a ne greška ili izuzetno stanje.

3. AST (Abstract Syntax Tree)

   U klasi Parser pravimo AST tako sto pravimo cvorove i to dvije vrste cvorova NodeExpression i NodeStatement. Kada napisemo neki upit za bazu pravi se AST i tako npr. ako unesemo CREATE TABLE table_name stablo ce izgledati ovako:

   NodeCreateTable
   |
   |
   table_name

4. Upotreba user autentifikacije sa enkripcijom sifre

   U klasi User citamo i upisujemo u fajl users.json na sljedeci nacin.

   Kada se pravi novi user sifra se enkriptuje na jednostavan nacin tako sto se se svaki znak zamjenjuje sljedećim znakom u definiranom skupu znakova (charset).

   Takodjer se generise randomiziran id ili cuid i pravi se folder sa tim id-om kao nazivom i u tom folderu ce se smjestati baze koje user pravi.

4. GitHub repozitorij

https://github.com/Inas1234/DBMS