МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**Практическая работа №1**

**«**Составление задач на алгоритмы**»**

**по дисциплине: «*Программирование*»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АИ-32», «АВТФ»  *Аталыков Иван*  *Александрович*  «5» мая 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  Доцент кафедры ЗИ  *Архипова Анастасия Борисовна*  «\_\_\_» \_\_\_\_\_\_ 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

**Цели и задачи работы:** у самурая нет цели, есть только путь.

**Задание к работе:** требуется реализовать реляционную СУБД с поддержкой запросов на языке SQL. СУБД использует единственный тип данных - строка.

Требуется поддержать следующие SQL выражения:

1. SELECT <> FROM <> - выборка

2. WHERE и операторы OR , AND – фильтрация

3. INSERT INTO - вставка данных в таблицы

4. DELETE FROM - удаление данных из таблицы

Выполнения данного здания я буду проводить на языке GO.

В папке нашей программы будет находится два файла:

1. Main.go -  основной файл для работы с базой данных.
2. Schema.json – файл разметки csv файлов

Код файла scheme.json

{

"name": "my\_database",

"tuples\_limit": 1000,

"structure": {

"таблица1": ["id", "name", "age", "email"],

"таблица2": ["product\_id", "product\_name", "price"]

}

}

В данным файле мы делаем разметку двух будущих CSV таблиц которые буду созданы кодом. Можно заметить максимальное кол-во строк т.е 1000 и заголовки столбцов двух таблиц.

Код фалйа main.go:

Импортируем нужные пакеты для корректной работы программы

package main

import (

"bufio"

"encoding/csv"

"encoding/json"

"fmt"

"os"

"strings"

)

Далее импортируем данные из ране упомянутого файла scheme.json

type Schema struct {

Name string `json:"name"`

TuplesLimit int `json:"tuples\_limit"`

Structure map[string][]string `json:"structure"`

}

func main() {

// Чтение схемы из файла

file, err := os.Open("scheme.json")

if err != nil {

fmt.Println("Ошибка при открытии файла:", err)

return

}

defer file.Close()

var schema Schema

if err := json.NewDecoder(file).Decode(&schema); err != nil {

fmt.Println("Ошибка при декодировании JSON:", err)

return

}

Сделаем примитивный интерфейс для работы с программой.

// Интерфейс командной строки

fmt.Println("Введите команды для управления таблицами (или 'exit' для выхода):")

var input string

for {

fmt.Print("> ")

// Чтение команды целиком (включая пробелы)

reader := bufio.NewReader(os.Stdin)

input, \_ = reader.ReadString('\n')

input = strings.TrimSpace(input)

if strings.ToLower(input) == "exit" {

break

}

processCommand(schema, input)

}

}

func processCommand(schema Schema, command string) {

parts := strings.Fields(command)

if len(parts) == 0 {

fmt.Println("Пожалуйста, введите команду.")

return

}

switch strings.ToUpper(parts[0]) {

case "INSERT":

if len(parts) < 5 {

fmt.Println("Использование: INSERT таблица значение1, значение2, значение3, значение4")

return

}

tableName := parts[1]

values := strings.Join(parts[2:], " ")

insertData(schema, tableName, values)

case "SELECT":

if len(parts) == 3 && strings.ToUpper(parts[1]) == "ALL" {

tableName := parts[2]

selectAllData(schema, tableName)

} else if len(parts) < 3 {

fmt.Println("Использование: SELECT таблица WHERE условие")

return

} else {

tableName := parts[1]

whereClause := strings.Join(parts[3:], " ")

selectData(schema, tableName, whereClause)

}

case "DELETE":

if len(parts) < 3 {

fmt.Println("Использование: DELETE FROM таблица WHERE условие")

return

}

tableName := parts[2]

whereClause := strings.Join(parts[4:], " ")

deleteData(schema, tableName, whereClause)

default:

fmt.Println("Неизвестная команда.")

}

}

Далее идет функция insert в которую мы заложим функцию проверки наличия файла, иначе его создания и проведения операций над ним

func insertData(schema Schema, tableName string, values string) {

if \_, exists := schema.Structure[tableName]; !exists {

fmt.Println("Таблица не существует:", tableName)

return

}

// Создаем или открываем файл в режиме добавления

file, err := os.OpenFile(tableName+".csv", os.O\_APPEND|os.O\_CREATE|os.O\_WRONLY, 0644)

if err != nil {

fmt.Println("Ошибка при открытии файла:", err)

return

}

defer file.Close()

// Создаем новый CSV writer

writer := csv.NewWriter(file)

// Разбиваем значения по запятой и записываем в файл

record := strings.Split(values, ", ")

err = writer.Write(record)

if err != nil {

fmt.Println("Ошибка при записи в файл:", err)

return

}

// Сохраняем запись

writer.Flush()

if err := writer.Error(); err != nil {

fmt.Println("Ошибка при сохранении записи:", err)

return

}

fmt.Println("Данные успешно вставлены.")

}

Реализации функции Select

func selectData(schema Schema, tableName string, whereClause string) {

if \_, exists := schema.Structure[tableName]; !exists {

fmt.Println("Таблица не существует:", tableName)

return

}

file, err := os.Open(tableName + ".csv")

if err != nil {

fmt.Println("Ошибка при открытии файла:", err)

return

}

defer file.Close()

reader := csv.NewReader(file)

records, err := reader.ReadAll()

if err != nil {

fmt.Println("Ошибка при чтении файла:", err)

return

}

var found bool

for \_, record := range records {

for \_, field := range record {

if strings.Contains(field, whereClause) {

fmt.Println(record)

found = true

break

}

}

}

if !found {

fmt.Println("Нет записей, соответствующих условию:", whereClause)

}

}

func selectAllData(schema Schema, tableName string) {

if \_, exists := schema.Structure[tableName]; !exists {

fmt.Println("Таблица не существует:", tableName)

return

}

file, err := os.Open(tableName + ".csv")

if err != nil {

fmt.Println("Ошибка при открытии файла:", err)

return

}

defer file.Close()

reader := csv.NewReader(file)

records, err := reader.ReadAll()

if err != nil {

fmt.Println("Ошибка при чтении файла:", err)

return

}

for \_, record := range records {

fmt.Println(record)

}

}

Функция Delete

func deleteData(schema Schema, tableName string, whereClause string) {

if \_, exists := schema.Structure[tableName]; !exists {

fmt.Println("Таблица не существует:", tableName)

return

}

file, err := os.ReadFile(tableName + ".csv")

if err != nil {

fmt.Println("Ошибка при чтении файла:", err)

return

}

records, err := csv.NewReader(strings.NewReader(string(file))).ReadAll()

if err != nil {

fmt.Println("Ошибка при чтении CSV:", err)

return

}

var updatedRecords [][]string

for \_, record := range records {

found := false

for \_, field := range record {

if strings.Contains(field, whereClause) {

found = true

break

}

}

if !found {

updatedRecords = append(updatedRecords, record)

}

}

// Записываем обновленные записи обратно в файл

fileOut, err := os.OpenFile(tableName+".csv", os.O\_WRONLY|os.O\_TRUNC|os.O\_CREATE, 0644)

if err != nil {

fmt.Println("Ошибка при открытии файла:", err)

return

}

defer fileOut.Close()

writer := csv.NewWriter(fileOut)

err = writer.WriteAll(updatedRecords)

if err != nil {

fmt.Println("Ошибка при записи в файл:", err)

return

}

writer.Flush()

if err := writer.Error(); err != nil {

fmt.Println("Ошибка при сохранении записи:", err)

return

}

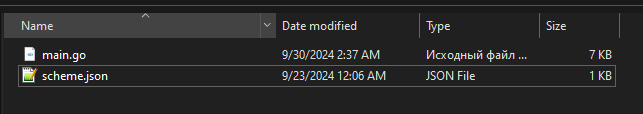
fmt.Println("Данные успешно удалены.")

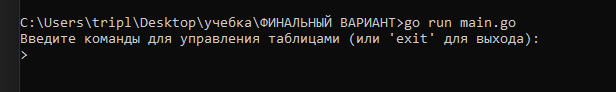
}

Конец….

Проведем испытание программы

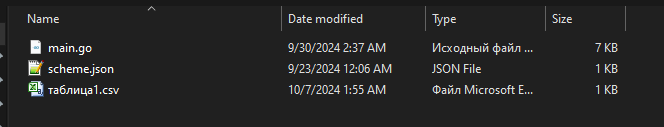
Изначальный вид папки программы:



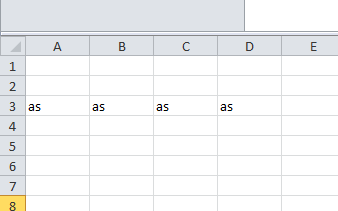
Запускаем код через терминал командой go run main.go. Нас встречает самый юзер френдли интерфейс..

Опробуем команду insert :  


после данной команды в папке создался файл таблица1.csv.



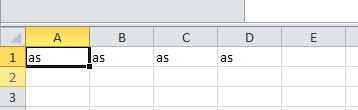
В котором благополучно появилась наша строка



(я два раза криво вписал строку пришлось удалять простите не бейте плетью)

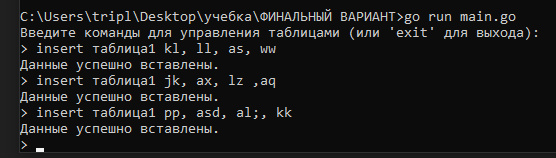
Повторим это с таблицей 2

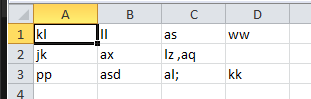




Протестируем команду delete:

Пересозадам программу и заполню ее по новой

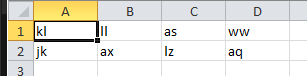




Например мне хочется удалит строку где есть упоминание сочетания букв kk.



В файле произойдет изменение и строка пропадет из файла

.

**Заключение**: данная практика позволила мне познакомиться с базами данных и порядком их создания, а также создания костылей.