Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Компьютерные системы и сети**

О Т Ч Е Т

по лабораторной работе № 8

Название: Организация клиент-серверного взаимодейсвтия между Golang и PostgreSQL

Дисциплина: языки интернет-программирования

Студент ИУ6-32Б _____ И.В.Порохницкий
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ В.Д.Шульман
(Подпись, дата) (И.О. Фамилия)

Москва, 2024

Цель работы - получение первичных навыков в организации долгосрочного хранения данных с использованием PostgreSQL и Golang

Задание

1. Установить и настроить PostgreSQL
2. Ознакомиться с теоретическими сведениями
3. Сделать форк данного репозитория в GitHub, склонировать получившуюся копию локально, создать от мастера ветку dev и переключиться на неё
4. Перекопировать код сервисов, полученный в ходе выполнения 6-й лабораторной работы, в соответствующие поддиректории в директории cmd (кроме кода сервиса hello, т.к. он уже реализован в качестве примера)
5. Доработать сервисы таким образом, чтобы они использовали для хранения данных СУБД PostgreSQL. Каждый сервис должен как добавлять новые данные в БД (insert/update), так и доставать их для предоставления пользователю (select)
6. Проверить свой код линтерами с помощью команды make lint
7. Сделать отчёт и поместить его в директорию docs
8. Зафиксировать изменения, сделать коммит и отправить получившееся состояние ветки дев в личный форк данного репозитория в GitHub
9. Через интерфейс GitHub создать Pull Request dev --> master

Ход работы

1. Установил и настроил PostgreSQL
2. Ознакомился с теоретическими сведениями
3. Сделал форк данного репозитория в GitHub, склонировал получившуюся копию локально, создал от мастера ветку дев и переключился на нее
4. Выполнил 3 задания:

   Задание №1 “count”:
   Необходимо написать веб сервер, аналогичный данному из 6-ой лабораторной работы:
   Напиши веб сервер (**порт :3333**) - счетчик который будет обрабатывать GET (/count) и POST (/count) запросы:
   GET: возвращает счетчик
   POST: увеличивает ваш счетчик на значение (с ключом "count") которое вы получаете из формы, но если пришло НЕ число то нужно ответить клиенту: "это не число" со статусом http.StatusBadRequest (400).

В ходе выполнения программы была построена таблица count в БД sandbox (рис. 1):

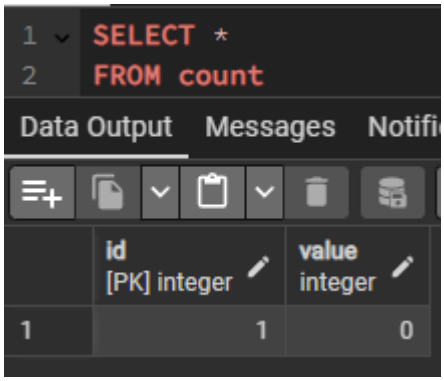

Рис.1

Таблица может как изначально присутствовать, так создаваться автоматически, из поля value по get-запросу берётся значение или изменяется при post-запросе. Результат работы представлен на Рис.2:

```go
package main

import (
    "database/sql"
    "fmt"
    "log"
    "net/http"
    "strconv"

    _ "github.com/lib/pq"
)

const (
    host     = "localhost"
    port     = 5432
    user     = "postgres"
    password = "BKMZ661248082"
    dbname   = "Sandbox"
)

type Handlers struct {
    dbProvider DatabaseProvider
}

type DatabaseProvider struct {
    db *sql.DB
}

func (h *Handlers) postCount(w http.ResponseWriter, r *http.Request) {
    r.ParseForm()
    val, err := strconv.Atoi(r.FormValue("count"))
    if err != nil {
        http.Error(w, "IT'S NOT NUMBER: "+err.Error(), http.StatusBadRequest)
        return
    }
    err = h.dbProvider.incrementCount(val)
    if err != nil {
        http.Error(w, "Database error: "+err.Error(), http.StatusInternalServerError)
        return
    }
    w.WriteHeader(http.StatusOK)
    w.Write([]byte("введено значение " + strconv.Itoa(val)))
}

func (h *Handlers) GetCount(w http.ResponseWriter, r *http.Request) {
    val, err := h.dbProvider.SelectCount()
    if err != nil {
        http.Error(w, "Database error: "+err.Error(), http.StatusInternalServerError)
        return
    }
    w.WriteHeader(http.StatusOK)
    w.Write([]byte("получено значение " + strconv.Itoa(val)))
}

func (dp *DatabaseProvider) SelectCount() (int, error) {
    var val int
    err := dp.db.QueryRow("SELECT value FROM count WHERE id = 1").Scan(&val)
    if err != nil {
        return 0, err
    }
    return val, nil
}

func (dp *DatabaseProvider) incrementCount(n int) error {
    _, err := dp.db.Exec("UPDATE count SET value = value + ($1) WHERE id = 1", n)
    return err
}

func main() {
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable", host, port, user, password, dbname)
    db, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    _, err = db.Exec(`
    CREATE TABLE IF NOT EXISTS count (
        id INTEGER PRIMARY KEY,
        value INTEGER NOT NULL DEFAULT 0
    )
    `)
    if err != nil {
        log.Fatal(err)
    }

    _, err = db.Exec("INSERT INTO count (id, value) VALUES (1, 0) ON CONFLICT (id) DO NOTHING")
    if err != nil {
        log.Fatal(err)
    }

    dp := DatabaseProvider{db: db}
    h := Handlers{dbProvider: dp}

    http.HandleFunc("/count", func(w http.ResponseWriter, r *http.Request) {
        switch r.Method {
        case "GET":
            h.GetCount(w, r)
        case "POST":
            h.postCount(w, r)
        default:
            http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        }
    })

    if err := http.ListenAndServe(":3333", nil); err != nil {
        log.Fatal(err)
    }
}
```

Командная строка

```
Microsoft Windows [Version 10.0.19045.5247]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\asset>curl -X GET http://localhost:3333/count
получено значение 0
C:\Users\asset>curl -X POST -d "count=-5" http://localhost:3333/count
введено значение -5
C:\Users\asset>curl -X GET http://localhost:3333/count
получено значение -5
C:\Users\asset>curl -X POST -d "count=17" http://localhost:3333/count
введено значение 17
C:\Users\asset>curl -X GET http://localhost:3333/count
получено значение 12
C:\Users\asset>
```

```
1  SELECT *
2  FROM count
```

Data Output Messages Notific

	id [PK] integer	value integer
1	1	12

Рис.2

Задание №2 “query”(Рис. 3):

Напишите веб сервер, который по пути /get отдает текст "Hello, web!". Порт должен быть :8080.
В данной реализации сервер выдаёт случайное сообщение из БД, передающееся на сервер JSON-ом

```go
cmd > hello > main.go > ...
package main

import (
    "database/sql"
    "encoding/json"
    "fmt"
    "log"
    "net/http"

    _ "github.com/lib/pq"
)

const (
    host     = "localhost"
    port     = 5432
    user     = "postgres"
    password = "BKMZ661248082"
    dbname   = "sandbox"
)

type Handlers struct {
    dbProvider DatabaseProvider
}

type DatabaseProvider struct {
    db *sql.DB
}

// Обработчики HTTP-запросов
func (h *Handlers) GetHello(w http.ResponseWriter, r *http.Request) {
    msg, err := h.dbProvider.SelectHello()
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
    }

    w.WriteHeader(http.StatusOK)
    w.Write([]byte(msg))
}
func (h *Handlers) PostHello(w http.ResponseWriter, r *http.Request) {
    input := struct {
        Msg string `json:"msg"`
    }{}

    decoder := json.NewDecoder(r.Body)
    err := decoder.Decode(&input)
    if err != nil {
        if err != nil {
            w.WriteHeader(http.StatusBadRequest)
            w.Write([]byte(err.Error()))
        }
    }

    err = h.dbProvider.InsertHello(input.Msg)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
    }

    w.WriteHeader(http.StatusCreated)
}

// Методы для работы с базой данных
func (dp *DatabaseProvider) SelectHello() (string, error) {
    var msg string

    // Получаем одно сообщение из таблицы hello, отсортированной в случайном порядке
    row := dp.db.QueryRow("SELECT message FROM hello ORDER BY RANDOM() LIMIT 1")
    err := row.Scan(&msg)
    if err != nil {
        return "", err
    }

    return msg, nil
}
func (dp *DatabaseProvider) InsertHello(msg string) error {
    _, err := dp.db.Exec("INSERT INTO hello (message) VALUES ($1)", msg)
    if err != nil {
        return err
    }

    return nil
}

func main() {

    // Формирование строки подключения для postgres
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
        "password=%s dbname=%s sslmode=disable",
        host, port, user, password, dbname)

    // Создание соединения с сервером postgres
    db, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    // Создаем провайдер для БД с набором методов
    dp := DatabaseProvider{db: db}
    // Создаем экземпляр структуры с набором обработчиков
    h := Handlers{dbProvider: dp}

    // Регистрируем обработчики
    http.HandleFunc("/get", h.GetHello)
    http.HandleFunc("/post", h.PostHello)

    // Запускаем веб-сервер на указанном адресе
    err = http.ListenAndServe(":8081", nil)
    if err != nil {
        log.Fatal(err)
    }
}
```

Рис.3

Результат работы (рис. 4):

```
mbi — -zsh — 79×24
Last login: Tue Dec 17 00:51:57 on ttys010
mbi@MacBook-Air-ila ~ % curl -X POST -H "Content-Type: application/json" -d '{"
msg":"Привет, Go!"}' http://localhost:8081/post
mbi@MacBook-Air-ila ~ % curl -X GET http://localhost:8081/get
Привет, Go!%
mbi@MacBook-Air-ila ~ %
```

рис. 4

В ходе выполнения программы были внесены изменения в таблицу hello в БД sandbox (рис. 5):

```
1  SELECT *
2  FROM hello
```

Data Output Messages Notifications

	id [PK] integer	message character varying (100)
1	1	Привет, мир!
2	2	Привет, мир!
3	3	Привет, Go!

Рис.5

Задание №2 “query”(Рис. 6):

Напишите веб-сервер который по пути /api/user приветствует пользователя:
Принимает и парсит параметр name и делает ответ "Hello,<name>!"
Пример: /api/user?name=Golang
Ответ: Hello,Golang!
В данной реализации по post-запросу в БД будет поступать name,
По get-запросу будет выводиться сообщение, при наличии на БД данного имени

cmd > query > main.go > (*DatabaseProvider).SelectName

```go
package main

import (
    "database/sql"
    "fmt"
    "log"
    "net/http"

    _ "github.com/lib/pq"
)

const (
    host     = "localhost"
    port     = 5432
    user     = "postgres"
    password = "BKMZ661248082"
    dbname   = "Sandbox"
)

type Handlers struct {
    dbProvider DatabaseProvider
}

type DatabaseProvider struct {
    db *sql.DB
}

func (h *Handlers) postQuery(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query().Get("name")
    if name == "" {
        http.Error(w, "IT'S EMPTY: ", http.StatusBadRequest)
        return
    }
    err := h.dbProvider.insertName(name)
    if err != nil {
        http.Error(w, "Database error: "+err.Error(), http.StatusInternalServerError)
        return
    }
    w.WriteHeader(http.StatusOK)
    w.Write([]byte("name is post"))
}

func (h *Handlers) GetQuery(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query().Get("name")
    if name == "" {
        fmt.Fprintf(w,"name is empty")
        return
    }
    err := h.dbProvider.SelectName(name)
    if err != nil {
        fmt.Fprint(w, "user does not exist!")
        return
    }
    w.WriteHeader(http.StatusOK)
    w.Write([]byte("Hello, " + name + "!"))

}

func (dp *DatabaseProvider) SelectName(name string) error {
    var exist string
    err := dp.db.QueryRow("SELECT name FROM query WHERE name = ($1)",name).Scan(&exist)
    if err != nil {
        return err
    }
    return nil
}

func (dp *DatabaseProvider) insertName(name string) error {
    _, err := dp.db.Exec("INSERT INTO query (name) VALUES ($1) ", name)
    return err
}

func main() {
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable", host, port, user, password, dbname)
    db, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    _, err = db.Exec(`
    CREATE TABLE IF NOT EXISTS query (
        id SERIAL PRIMARY KEY,
        name VARCHAR(50)
    )
    `)
    if err != nil {
        log.Fatal(err)
    }
    dp := DatabaseProvider{db: db}
    h := Handlers{dbProvider: dp}

    http.HandleFunc("/api/user/get", h.GetQuery)
    http.HandleFunc("/api/user/post", h.postQuery)

    if err := http.ListenAndServe(":9000", nil); err != nil{
        log.Println("Серверная ошибка:",err)
    }
}
```

рис. 6

Результат работы (рис. 7):

```
mbi — -zsh — 79×24
Last login: Tue Dec 17 00:54:13 on ttys022
mbi@MacBook-Air-ila ~ % curl -X GET "http://localhost:9000/api/user/get?name=Ivan"
user does not exist!%
mbi@MacBook-Air-ila ~ % curl -X POST "http://localhost:9000/api/user/post?name=Ivan"
name is post%
mbi@MacBook-Air-ila ~ % curl -X GET "http://localhost:9000/api/user/get?name=Ivan"
Hello, Ivan!%
mbi@MacBook-Air-ila ~ %
```

рис. 7

В ходе выполнения программы были внесены изменения в таблицу hello в БД sandbox (рис. 8):

```
1  SELECT *
2  FROM query
```

Data Output Messages Notifications

	id [PK] integer	name character varying (50)
1	1	Ivan

Рис.8

5. Сделал отчёт и поместил его в директорию docs
Зафиксировал изменения, сделал коммит и отправил полученное состояние ветки дев в удаленный репозиторий GitHub

Вывод

Я получил первичные навыки в организации долгосрочного хранения данных с использованием PostgreSQL и Golang.