



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 Информатика и вычислительная техника

О Т Ч Е Т

по лабораторной работе № 8

Название: Организация клиент-серверного взаимодействия между Golang и PostgreSQL

Дисциплина: Языки Интернет Программирования

Студент

ИУ6-32Б
(Группа)

(Подпись, дата)

А.С.Авдеева
(И.О. Фамилия)

Преподаватель

(Подпись, дата)

В.Д.Шульман
(И.О. Фамилия)

Москва, 2024

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

Задание - 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

1)Доработала сервис с Query – параметром

Код к приложению:

```
package main

import (
    "database/sql"
    "flag"
    "fmt"
    _ "github.com/lib/pq"
    "log"
    "net/http"
)

// Подключение к базе данных:
const (
    host      = "localhost"
    port      = 5432
    user      = "postgres"
    password  = "123"
    dbname    = "db1_query"
)

// Обработчики HTTP запросов:
type Handlers struct {
    dbProvider DatabaseProvider
}

// структура для подключения к Базе Данных:
type DatabaseProvider struct {
    db *sql.DB
}

// Методы для HTTP запросов:
func (h *Handlers) GetQuery(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query().Get("name")
    if name == "" {
        w.WriteHeader(http.StatusBadRequest)
        w.Write([]byte("Введите имя с кем здороваться!"))
        return
    }
    name, err := h.dbProvider.SelectQuery(name)
    if err != nil {
        if err == sql.ErrNoRows {
            w.WriteHeader(http.StatusInternalServerError)
            w.Write([]byte("Такого человека нет в базе данных"))
            return
        }
    }
}
```

```

        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
        return

    }

    w.WriteHeader(http.StatusOK)
    w.Write([]byte("Hello," + " " + nam + "!"))
    return
}

func (h *Handlers) PostQuery(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query().Get("name")
    err := h.dbProvider.InsertQuery(name)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
        return
    }
    w.WriteHeader(http.StatusCreated)
}

// Методы для работы с БД:
func (dp *DatabaseProvider) SelectQuery(name string) (string, error) {
    var nam string
    row := dp.db.QueryRow("SELECT name FROM table1 WHERE name = ($1)", name)
    err := row.Scan(&nam)
    if err != nil {
        return "", err
    }
    return nam, nil
}

func (dp *DatabaseProvider) InsertQuery(name string) error {
    _, err := dp.db.Exec("INSERT INTO table1 (name) VALUES ($1)", name)

    if err != nil {
        return err
    }
    return nil
}

func main() {
    // Считываем аргументы командной строки
    address := flag.String("address", "127.0.0.1:9000", "адрес для запуска сервера")
    flag.Parse()

    // 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

```

```

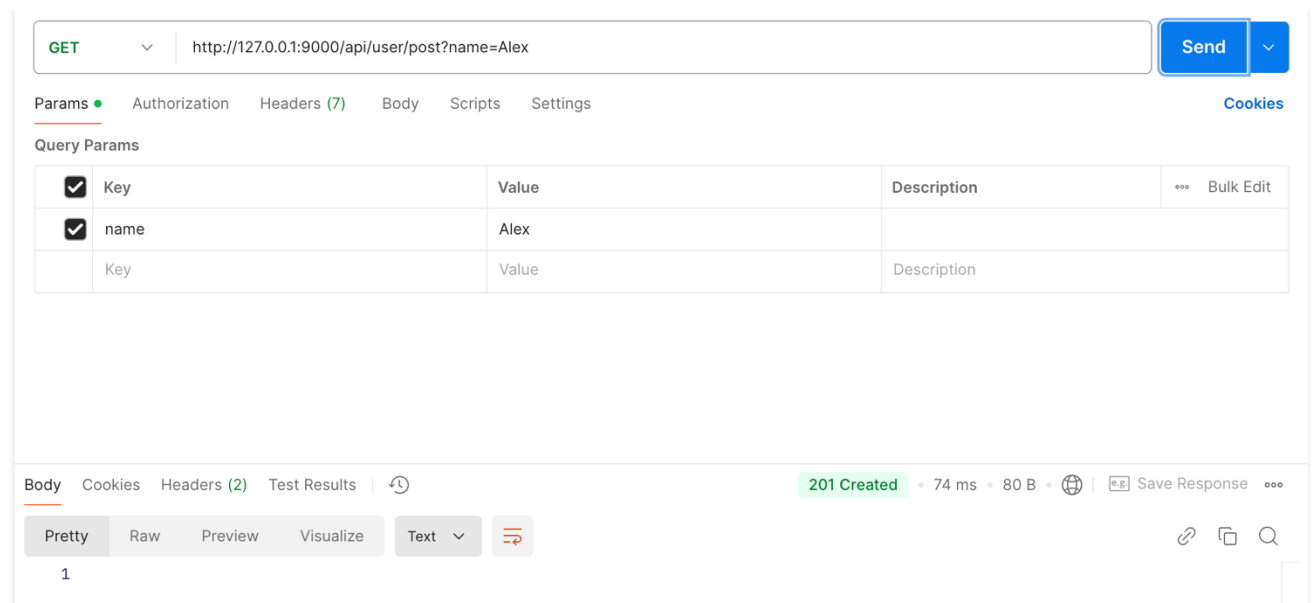
http.HandleFunc("/api/user/get", h.GetQuery)
http.HandleFunc("/api/user/post", h.PostQuery)

// Запускаем веб-сервер на указанном адресе
err = http.ListenAndServe(*address, nil)
if err != nil {
    log.Fatal(err)
}
}

```

Как работает:

1)Добавим в базу данных имя Alex(рис.1):



(рис.1)

2)Смотрим что имя Alex добавилось в нашу таблицу через pgAdmin4(рис.2):

Query

Query History

1

2

SELECT * FROM public.table1

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑

🗄

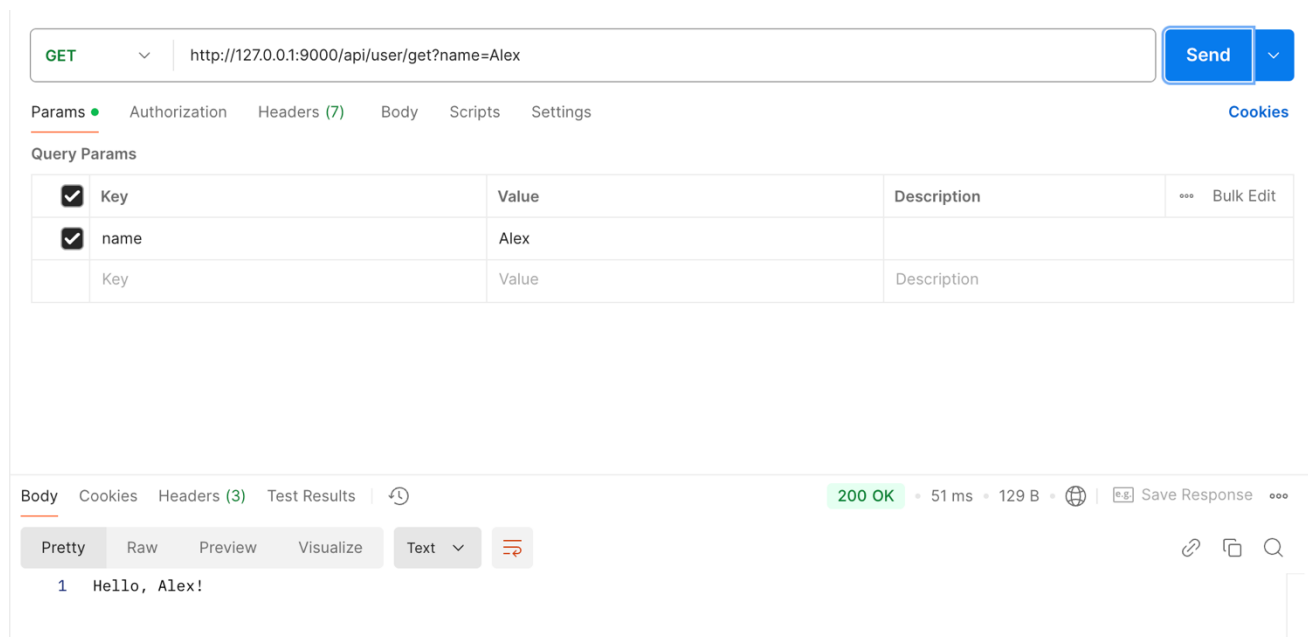
⬇

📈

| | name text | 🔒 |
|---|--------------|---|
| 1 | Alena | |
| 2 | Alex | |

(рис.2)-Имя Alex добавлено

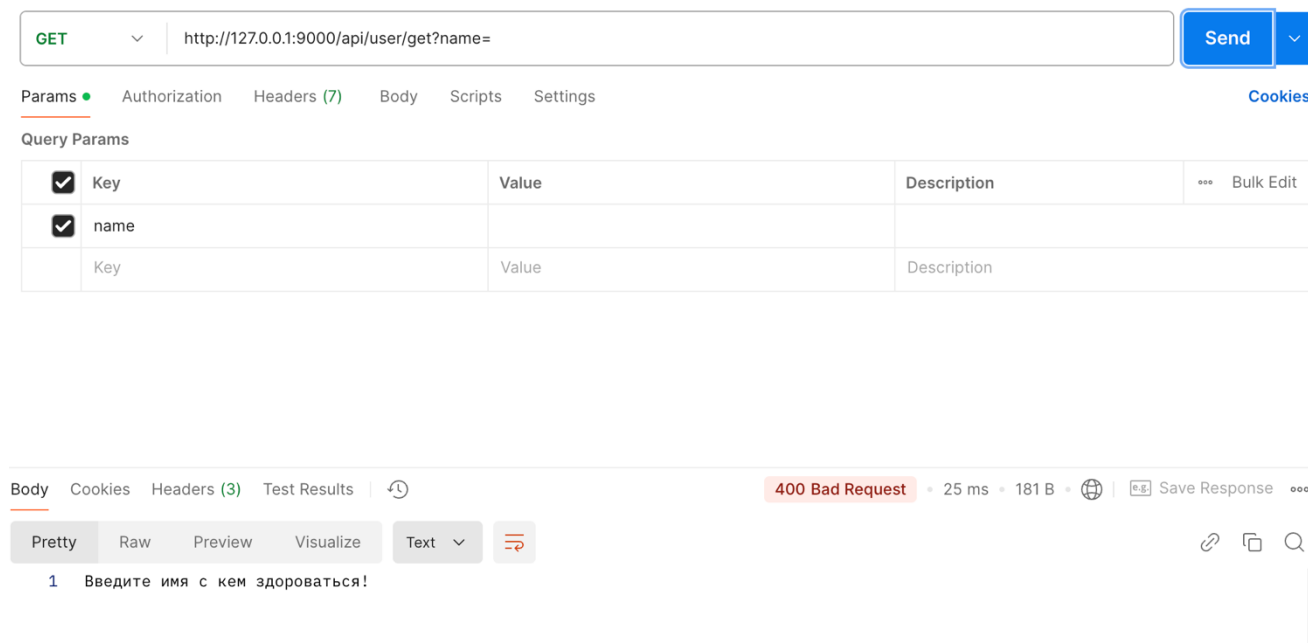
3)Выведем имя Alex, чтобы он поздоровался с ним(рис.3):



(рис.3)

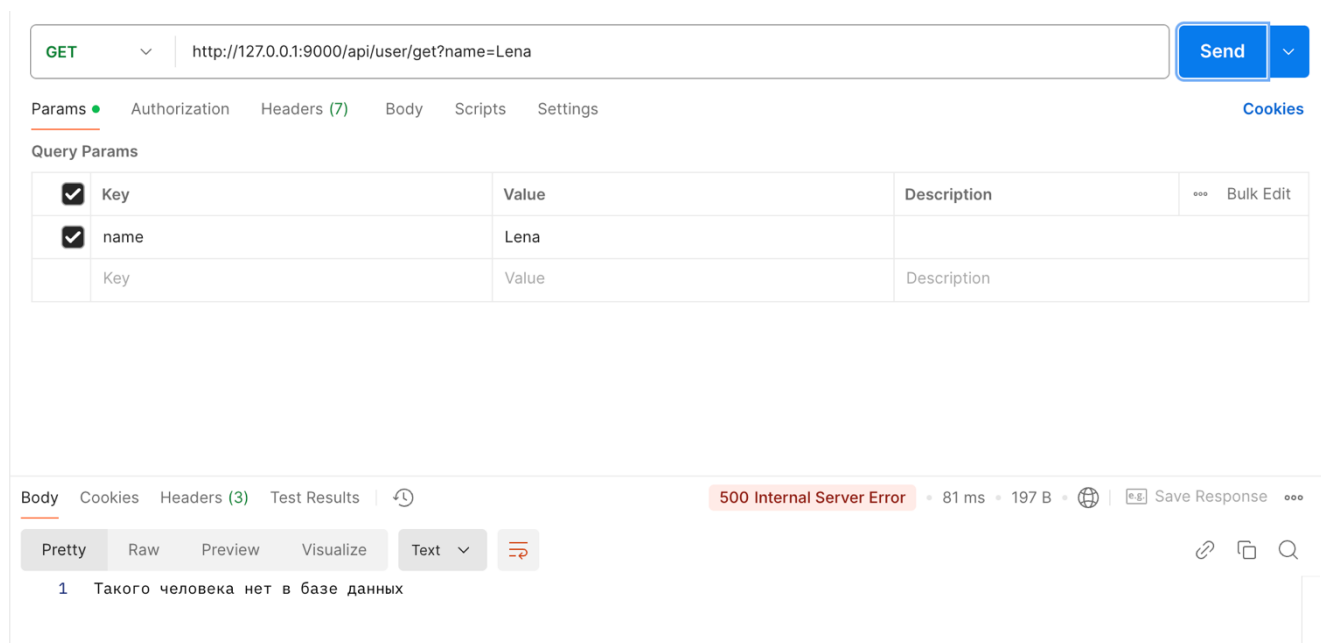
Проверим работу ошибок, прописанных в коде:

1) На ввод пустой строки (рис.4):



(рис.4)

2) На ввод человека, которого нет в базе данных (рис.5):



(рис. 5)

2) Задание на count:

Код к приложению:

```
package main

import (
    "database/sql"
    "flag"
    "fmt"
    _ "github.com/lib/pq"
    "log"
    "net/http"
    "strconv"
)

// Подключение к базе данных:
const (
    host      = "localhost"
    port      = 5432
    user      = "postgres"
    password  = "123"
    dbname    = "db2_count"
)

// Обработчики HTTP запросов:
type Handlers struct {
    dbProvider DatabaseProvider
}

// структура для подключения к Базе Данных:
type DatabaseProvider struct {
    db *sql.DB
}
```

```
// Методы для HTTP запросов:
func (h *Handlers) GetCount(w http.ResponseWriter, r *http.Request) {
    if r.Method == "GET" {
        count, err := h.dbProvider.SelectQuery()
        num := count
        if err != nil {
            w.WriteHeader(http.StatusInternalServerError)
            w.Write([]byte(err.Error()))
            return
        }
        w.WriteHeader(http.StatusOK)
        w.Write([]byte(strconv.Itoa(num)))
    } else {
        h.PostCount(w, r)
    }
}

func (h *Handlers) PostCount(w http.ResponseWriter, r *http.Request) {
    r.ParseForm()
    num := r.Form.Get("count")
    count, err := strconv.Atoi(num)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        w.Write([]byte("Это не число!"))
        return
    }
    err = h.dbProvider.InsertQuery(count)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
        return
    }
    w.WriteHeader(http.StatusCreated)
    w.Write([]byte("Данные успешно добавлены в базу данных!"))
}

// Методы для работы с БД:
func (dp *DatabaseProvider) SelectQuery() (int, error) {
    var count int
    row := dp.db.QueryRow("SELECT count FROM table4")
    err := row.Scan(&count)
    if err != nil {
        return 0, err
    }
    return count, nil
}

func (dp *DatabaseProvider) InsertQuery(count int) error {
    _, err := dp.db.Exec("UPDATE table4 SET count = count + $1", count) // Changed
    // table and column name, using ON CONFLICT

    if err != nil {
        return err
    }
    return nil
}

func main() {
    // Считываем аргументы командной строки
    address := flag.String("address", ":3333", "адрес для запуска сервера")
    flag.Parse()

    // 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("/count", h.GetCount)

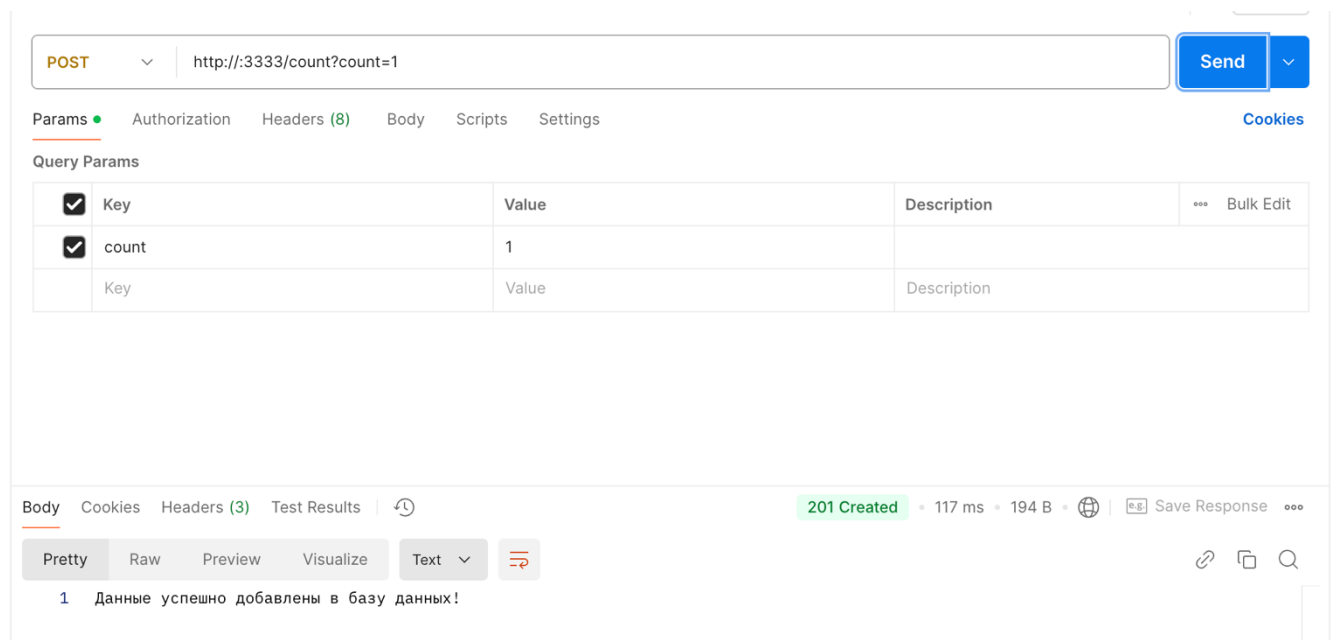
// Запускаем веб-сервер на указанном адресе
err = http.ListenAndServe(*address, nil)
if err != nil {
    log.Fatal(err)
}
}

```

Как работает:

1)Так как у нас HTML форма, забьём значение count = 1 через

Postman(рис.1):



(рис.1)


2)Посмотрим, что данные были добавлены в базу данных через

PgAdmin4(рис.2):

Query Query History

```
1 SELECT * FROM table4
```

Data Output Messages Notifications

count integer 

| | |
|---|---|
| 1 | 1 |
|---|---|

(рис.2)-Данные успешно добавились!

3)Теперь выведем значение count = 1 через ручку get(рис.3):

GET http://:3333/count?count=1 Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

| <input checked="" type="checkbox"/> | Key | Value | Description | ... | Bulk Edit |
|-------------------------------------|-------|-------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> | count | 1 | | | |
| | Key | Value | Description | | |

Body Cookies Headers (3) Test Results 200 OK • 57 ms • 117 B Save Response

Pretty Raw Preview Visualize Text 1 1

(рис.3)

4)Добавим ещё значение 5 через ручку POST(рис.4):

POST http://:3333/count?count=5 Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Query Params

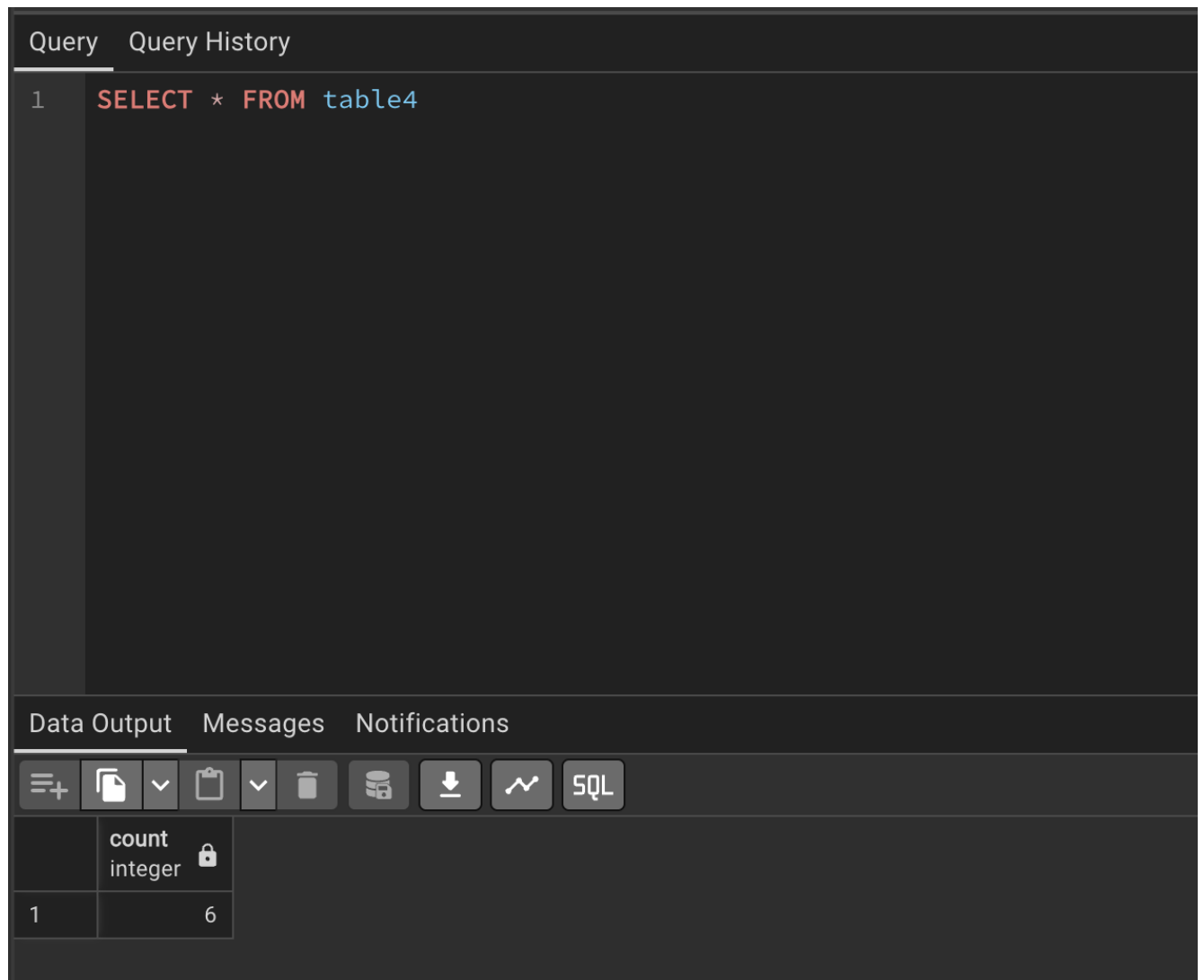
| <input checked="" type="checkbox"/> | Key | Value | Description | ... | Bulk Edit |
|-------------------------------------|-------|-------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> | count | 5 | | | |
| | Key | Value | Description | | |

Body Cookies Headers (3) Test Results 201 Created • 23 ms • 194 B Save Response

Pretty Raw Preview Visualize Text 1 Данные успешно добавлены в базу данных!

(рис.4)

5)Посмотрим, что сохранилось в базу данных(рис.5):



(рис. 5)-Всё верно, должна сохраняться сумма первого добавленного числа и второго

6)Посмотрим, что выведет Postman через ручку Get(рис. 6):

GET http://3333/count?count=5 Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

| <input checked="" type="checkbox"/> | Key | Value | Description | ... | Bulk Edit |
|-------------------------------------|-------|-------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> | count | 5 | | | |
| | Key | Value | Description | | |

Body Cookies Headers (3) Test Results 200 OK • 50 ms • 117 B Save Response

Pretty Raw Preview Visualize Text

1 6

(рис. 6)-Всё верно!

7)Проверим ошибки, прописанные в коде. Проверим, что будет, если введем не число, а строку (рис. 7):

POST http://3333/count?count=5hnh Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Query Params

| <input checked="" type="checkbox"/> | Key | Value | Description | ... | Bulk Edit |
|-------------------------------------|-------|-------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> | count | 5hnh | | | |
| | Key | Value | Description | | |

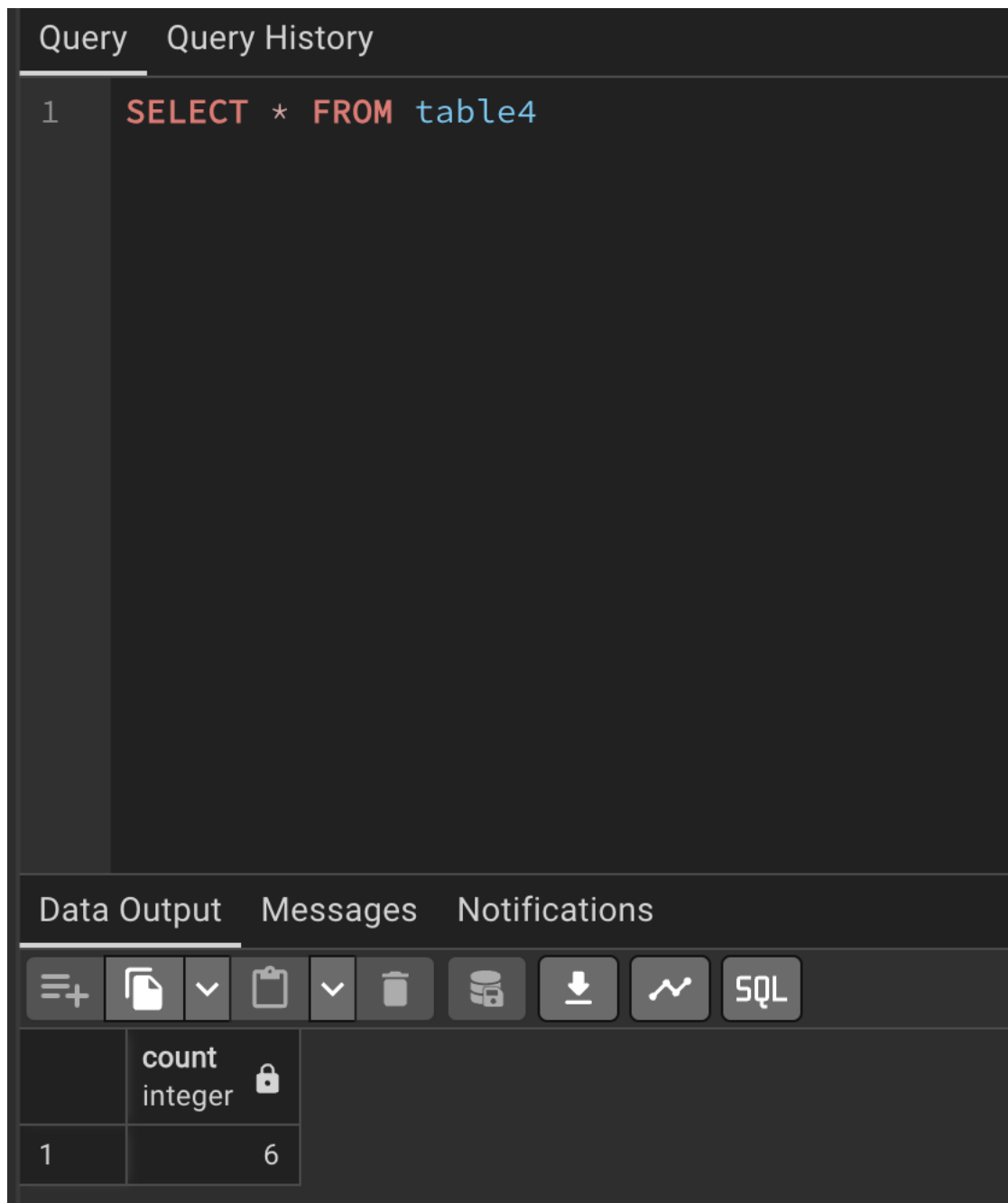
Body Cookies Headers (3) Test Results 400 Bad Request • 2 ms • 149 B Save Response

Pretty Raw Preview Visualize Text

1 Это не число!

(рис. 7)-Всё верно!

8)Проверим, что эта строка ни в коем случае не сохранилась в базу данных(рис.8):



(рис.8)-Всё верно! Всё работает!

3)Задание Hello – это задание прикреплено как пример

Код к приложению:

```
package main

import (
    "database/sql"
    "encoding/json"
    "flag"
    "fmt"
    "log"
```

```

    "net/http"

    _ "github.com/lib/pq"
)

const (
    host      = "localhost"
    port      = 5432
    user      = "postgres"
    password  = "123"
    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("Hello," + " " + 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() {
    // Считываем аргументы командной строки
    address := flag.String("address", "127.0.0.1:8081", "адрес для запуска сервера")
    flag.Parse()

    // 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}

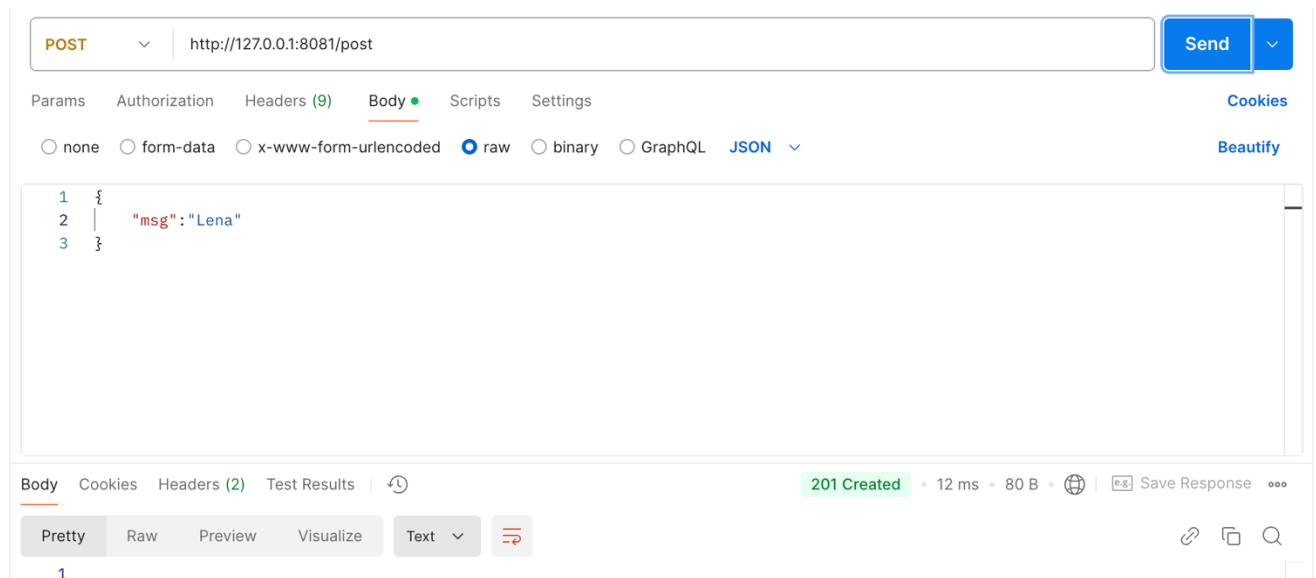
    // Регистрируем обработчики
    http.HandleFunc("/get", h.GetHello)
    http.HandleFunc("/post", h.PostHello)

    // Запускаем веб-сервер на указанном адресе
    err = http.ListenAndServe(*address, nil)
    if err != nil {
        log.Fatal(err)
    }
}

```

Как работает:

1)Я создала базу данных для работы этого сервера, проверим, что значения, вводящиеся через Postman добавляются в созданную базу данных(рис.1 и рис.2):



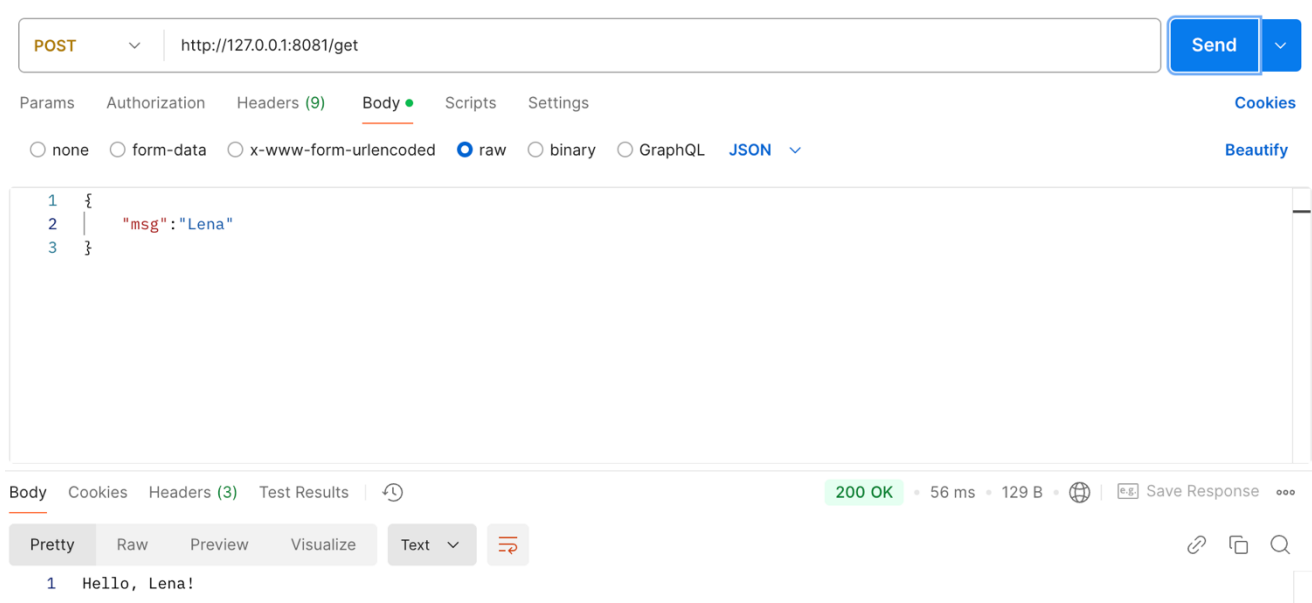
(рис.1)-добавили значение через Postman

The screenshot shows a database query editor interface. At the top, there are two tabs: "Query" and "Query History". The "Query" tab is active, displaying a SQL query: `SELECT * FROM public.hello`. Below the query editor, there are three tabs: "Data Output", "Messages", and "Notifications". The "Data Output" tab is active, showing a table of results. The table has two columns: an index column and a "message" column. The "message" column is labeled "message" and "text" with a lock icon. The results are as follows:

| | message text |
|---|-----------------|
| 1 | Alena |
| 2 | Lena |

(рис.2)-значение сохранилось в базу данных

2)Теперь через ручку get выведем приветствие(рис.3):



(рис.3)-вывели приветствие через get

Заключение – проделана успешная работа в понимании работы с Базой данных через Go!