

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

по лабораторной работе № 8

Дисциплина: Языки интернет-программирования

 (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

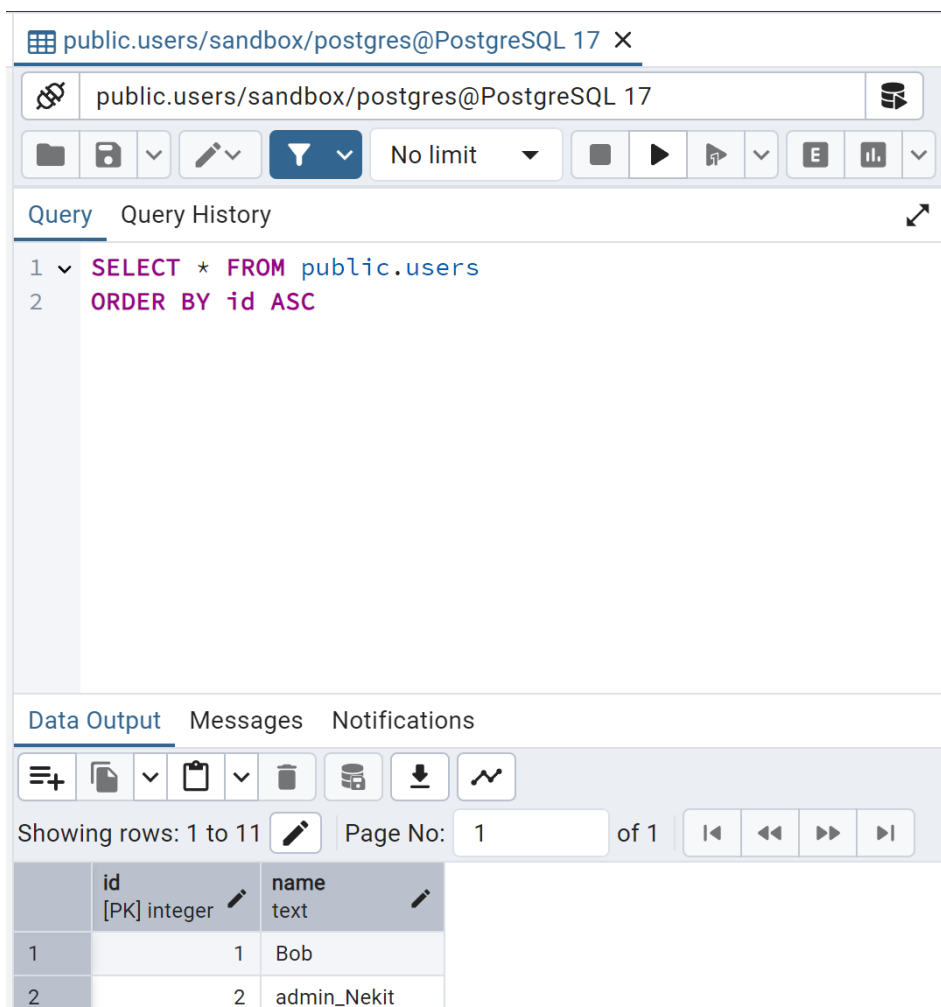
В рамках данной лабораторной работы предлагается продолжить изучение Golang и познакомиться с набором стандартных библиотек, используемых для организации клиент-серверного взаимодействия между Golang и Postgresql, где в роли клиента выступает сервис Golang, а в роли сервера СУБД Postgresql.

Ход работы:

1) Модифицируем код микросервиса query для сохранения имен(пользователей) в БД с ID.

Создаем таблицу:

```
CREATE TABLE users (  
    id SERIAL PRIMARY KEY,  
    name TEXT NOT NULL  
);
```



The screenshot shows a PostgreSQL query editor interface. The query entered is:

```
1 SELECT * FROM public.users  
2 ORDER BY id ASC
```

The results are displayed in a table with the following data:

	id [PK] integer	name text
1	1	Bob
2	2	admin_Nekit

Рис.1 — Таблица users в БД

Код программы:

```
package main

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

    _ "github.com/lib/pq"
)

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

type DatabaseProvider struct {
    db *sql.DB
}

type User struct {
    ID    int    `json:"id"`
    Name  string `json:"name"`
}

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

    // Проверяем соединение
    err = db.Ping()
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println("Successfully connected to the database!")

    // Инициализируем провайдер БД
    dbProvider := &DatabaseProvider{db: db}
```

```
// Регистрируем маршруты
http.HandleFunc("/api/user", func(w http.ResponseWriter, r *http.Request) {
    userHandler(w, r, dbProvider)
})
```

```
// Запускаем сервер
err = http.ListenAndServe(":9000", nil)
if err != nil {
    log.Fatal(err)
}
}
```

```
// Обработчик запросов
func userHandler(w http.ResponseWriter, r *http.Request, dbProvider
*DatabaseProvider) {
    w.Header().Set("Access-Control-Allow-Origin", "*")
    w.Header().Set("Access-Control-Allow-Methods", "GET, POST, OPTIONS")
    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")
```

```
    switch r.Method {
    case http.MethodGet:
        // Получение информации о пользователе по имени
        name := r.URL.Query().Get("name")
        if name == "" {
            http.Error(w, "Parameter 'name' is required", http.StatusBadRequest)
            return
        }
    }
```

```
    user, err := dbProvider.GetUser(name)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
```

```
    if user == nil {
        http.Error(w, "User not found", http.StatusNotFound)
        return
    }
```

```
    if err := json.NewEncoder(w).Encode(user); err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
```

```
    case http.MethodPost:
        // Добавление нового пользователя
        var user User
        err := json.NewDecoder(r.Body).Decode(&user)
        if err != nil {
            http.Error(w, "Invalid JSON format", http.StatusBadRequest)
            return
        }
    }
```

```

    }

    err = dbProvider.AddUser(user.Name)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    w.WriteHeader(http.StatusCreated)
    fmt.Fprintf(w, "User %s added successfully", user.Name)

    default:
        http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
    }
}

// Методы работы с базой данных
func (dp *DatabaseProvider) GetUser(name string) (*User, error) {
    query := "SELECT id, name FROM users WHERE name = $1"
    row := dp.db.QueryRow(query, name)

    var user User
    err := row.Scan(&user.ID, &user.Name)
    if err == sql.ErrNoRows {
        return nil, nil // Пользователь не найден
    } else if err != nil {
        return nil, err
    }

    return &user, nil
}

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

```

Результат работы:

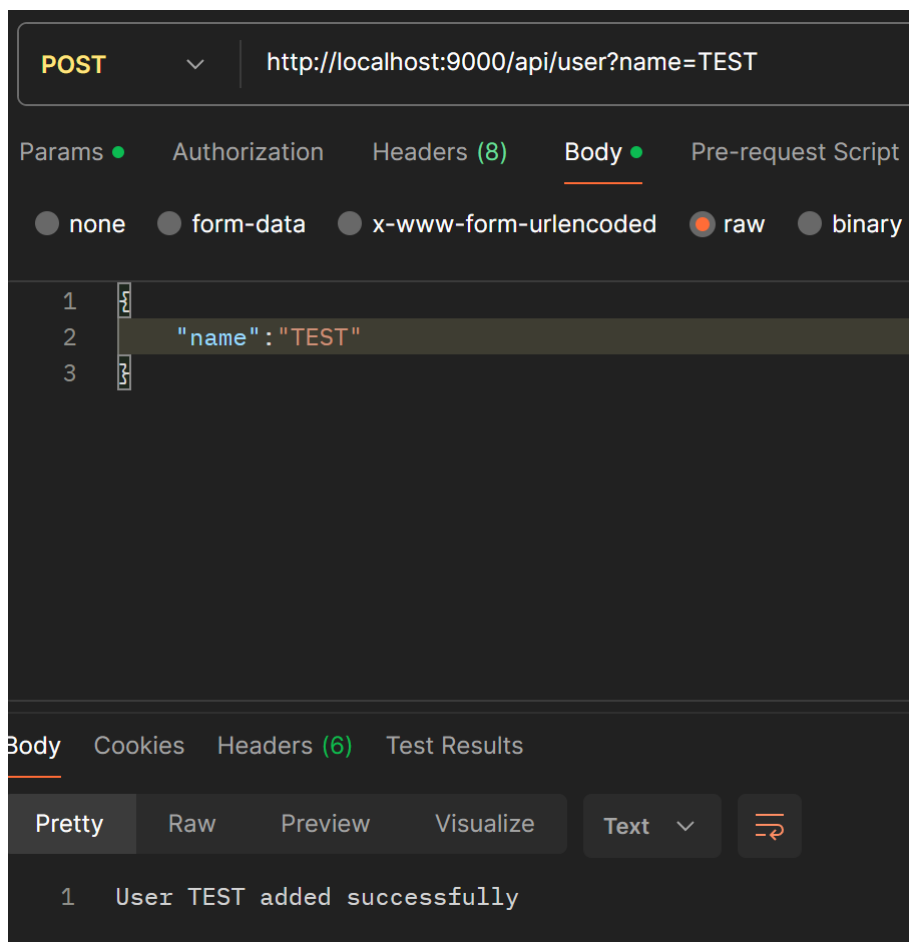


Рис.2 — Добавление пользователя

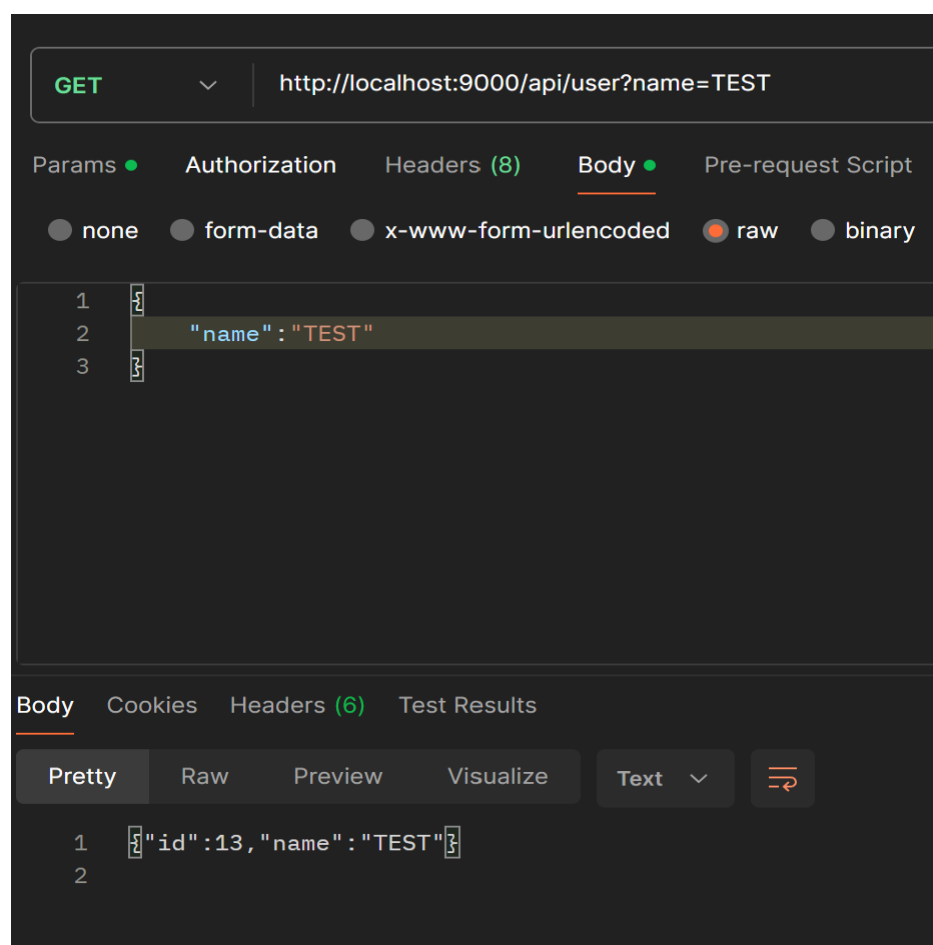


Рис.3 — Получение пользователя по имени из БД

2) Модифицируем код микросервиса count для хранения значения счетчика в БД.

Создание таблицы в БД:

```
CREATE TABLE counter (  
    id SERIAL PRIMARY KEY,  
    value INT NOT NULL  
);
```

Код программы:

```
package main  
  
import (  
    "database/sql"  
    "encoding/json"  
    "fmt"  
    "log"  
    "net/http"  
    "strconv"  
  
    _ "github.com/lib/pq"  
)  
  
const (  
    host      = "localhost"  
    port      = 5432  
    user      = "postgres"  
    password  = "postgres"  
    dbname    = "sandbox"  
)  
  
type DatabaseProvider struct {  
    db *sql.DB  
}  
  
type Counter struct {  
    ID      int `json:"id"`  
    Value   int `json:"value"`  
}  
  
func main() {  
    // Подключение к PostgreSQL  
    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()
```

```

// Проверяем соединение
err = db.Ping()
if err != nil {
    log.Fatal(err)
}
fmt.Println("Successfully connected to the database!")

// Инициализируем провайдер БД
dbProvider := &DatabaseProvider{db: db}

// Добавляем начальное значение счетчика если оно отсутствует
err = dbProvider.initializeCounter()
if err != nil {
    log.Fatal(err)
}

// Регистрируем маршруты
http.HandleFunc("/count", func(w http.ResponseWriter, r *http.Request) {
    countHandler(w, r, dbProvider)
})

// Запускаем сервер
err = http.ListenAndServe(":3333", nil)
if err != nil {
    log.Fatal(err)
}
}

// Обработчик запросов
func countHandler(w http.ResponseWriter, r *http.Request, dbProvider
*DatabaseProvider) {
    w.Header().Set("Access-Control-Allow-Origin", "*")
    w.Header().Set("Access-Control-Allow-Methods", "GET, POST, OPTIONS")
    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")

    switch r.Method {
    case http.MethodGet:
        // Получение текущего значения счетчика
        counter, err := dbProvider.GetCounter()
        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }

        if counter == nil {
            http.Error(w, "Counter not found", http.StatusNotFound)
            return
        }

        if err := json.NewEncoder(w).Encode(counter); err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
        }
    }
}

```



```

    }

    case http.MethodPost:
        // Увеличение счетчика
        err := r.ParseForm()
        if err != nil {
            http.Error(w, "Invalid form data", http.StatusBadRequest)
            return
        }

        countStr := r.FormValue("count")
        count, err := strconv.Atoi(countStr)
        if err != nil {
            http.Error(w, "Count must be a number", http.StatusBadRequest)
            return
        }

        err = dbProvider.IncreaseCounter(count)
        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }

        fmt.Fprintf(w, "Counter increased by %d", count)

    default:
        http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
    }
}

// Методы работы с базой данных
func (dp *DatabaseProvider) GetCounter() (*Counter, error) {
    query := "SELECT id, value FROM counter LIMIT 1"
    row := dp.db.QueryRow(query)

    var counter Counter
    err := row.Scan(&counter.ID, &counter.Value)
    if err == sql.ErrNoRows {
        return nil, nil // Счетчик не найден
    } else if err != nil {
        return nil, err
    }

    return &counter, nil
}

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

```

```
// Инициализация счетчика, если он отсутствует
func (dp *DatabaseProvider) initializeCounter() error {
    var count Counter

    query := "SELECT id FROM counter LIMIT 1"
    err := dp.db.QueryRow(query).Scan(&count.ID)
    if err == sql.ErrNoRows {
        // Счетчик не найден, добавляем начальное значение
        insertQuery := "INSERT INTO counter (value) VALUES ($1)"
        _, err := dp.db.Exec(insertQuery, 0)
        if err != nil {
            return err
        }
    }

    return nil
}
```

Результат работы:

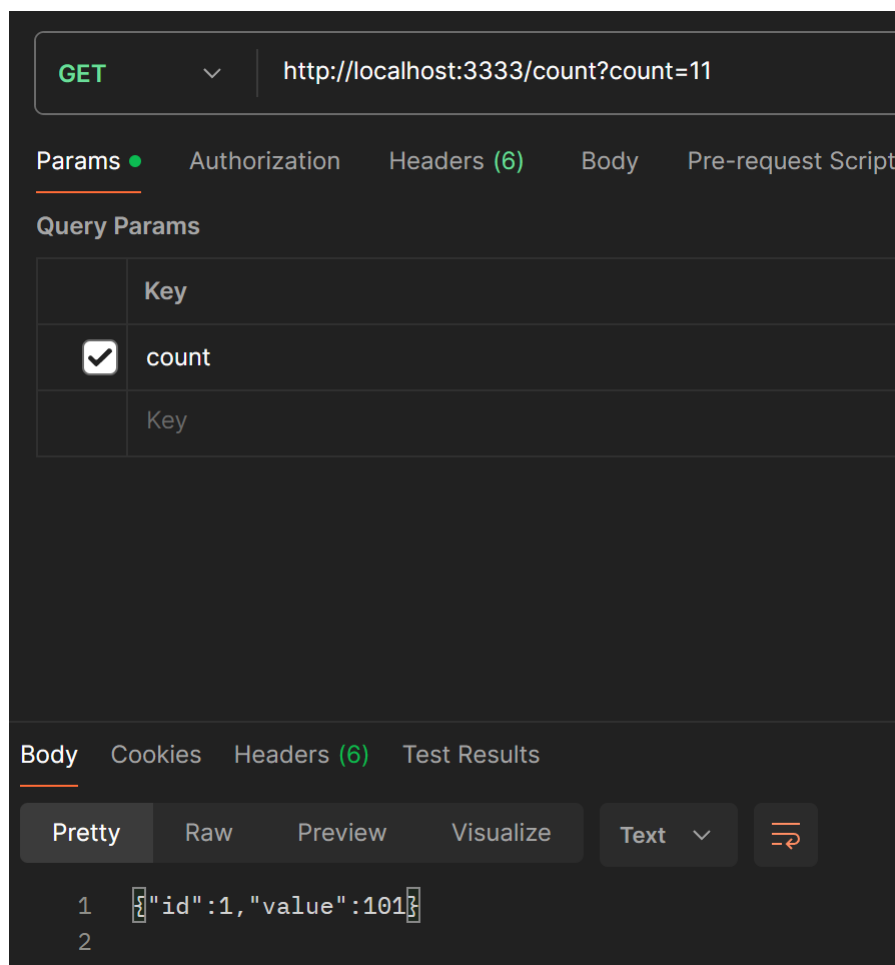


Рис.3 — Получение счетчика из БД

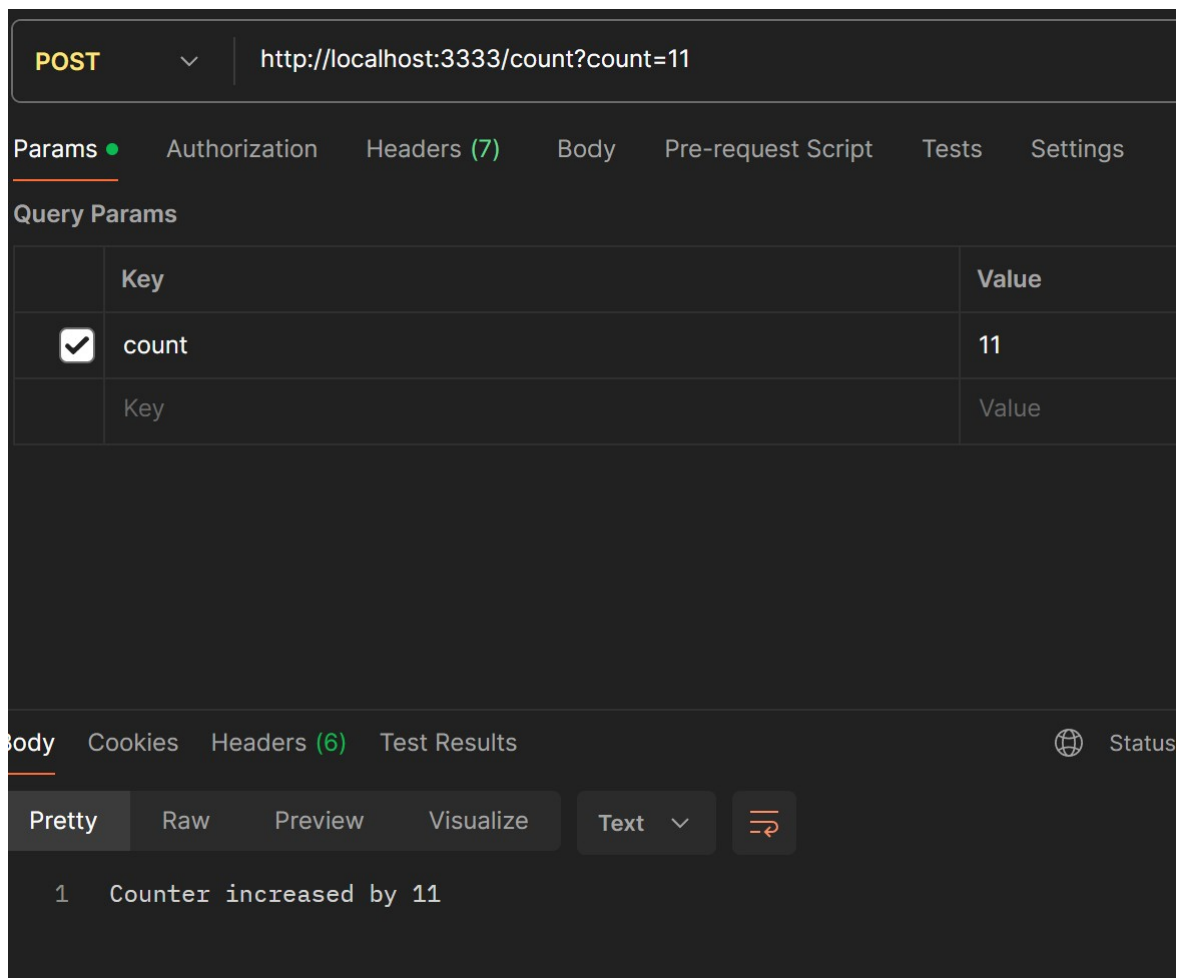


Рис.4 — Увеличение счетчика

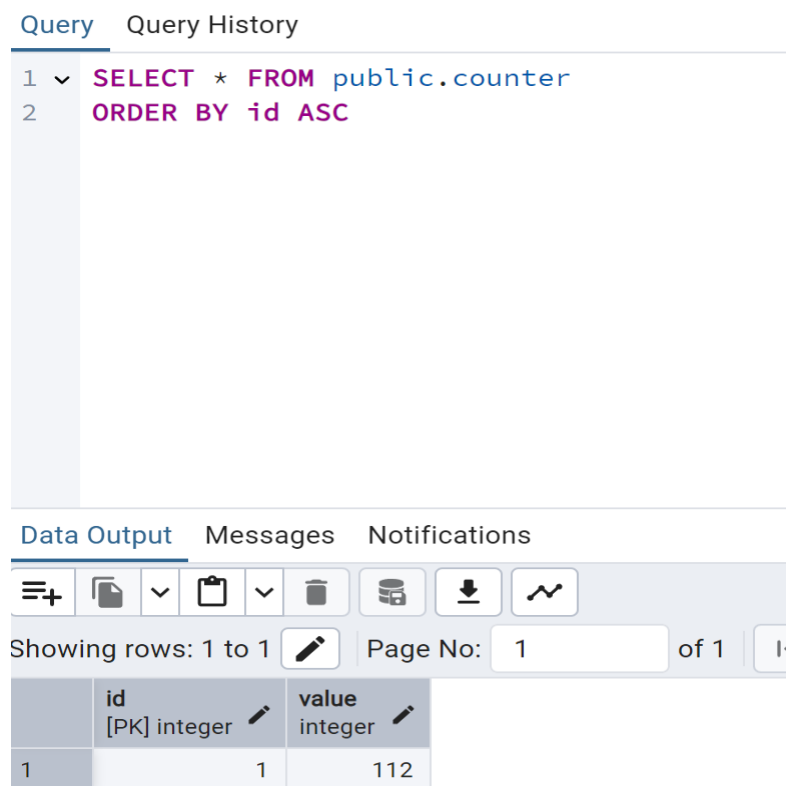


Рис.5 — Обновленный счетчика

Проверка линтерами:

```
PS C:\Users\1234\web-8> make lint
golangci-lint run
level=error msg="[linters_context] typechecking error: : # github.com/ValeryBMSU/web-8/cmd/count/ncmd\\count\\tempCodeRunnerFile.go:15:2: host redeclared in this block\\n\\tcmd\\count\\main.go:15:2: other declaration of host\\ncmd\\count\\tempCodeRunnerFile.go:16:2: port redeclared in this block\\n\\tcmd\\count\\main.go:16:2: other declaration of port\\ncmd\\count\\tempCodeRunnerFile.go:17:2: user redeclared in this block\\n\\tcmd\\count\\main.go:17:2: other declaration of user\\ncmd\\count\\tempCodeRunnerFile.go:18:2: password redeclared in this block\\n\\tcmd\\count\\main.go:18:2: other declaration of password\\ncmd\\count\\tempCodeRunnerFile.go:19:2: dbname redeclared in this block\\n\\tcmd\\count\\main.go:19:2: other declaration of dbname\\ncmd\\count\\tempCodeRunnerFile.go:22:6: DatabaseProvider redeclared in this block\\n\\tcmd\\count\\main.go:22:6: other declaration of DatabaseProvider\\ncmd\\count\\tempCodeRunnerFile.go:26:6: Counter redeclared in this block\\n\\tcmd\\count\\main.go:26:6: other declaration of Counter\\ncmd\\count\\tempCodeRunnerFile.go:31:6: main redeclared in this block\\n\\tcmd\\count\\main.go:31:6: other declaration of main\\ncmd\\count\\tempCodeRunnerFile.go:70:6: countHandler redeclared in this block\\n\\tcmd\\count\\main.go:70:6: other declaration of countHandler\\ncmd\\count\\tempCodeRunnerFile.go:121:29: method DatabaseProvider.GetCounter already declared at cmd\\count\\main.go:121:29\\ncmd\\count\\tempCodeRunnerFile.go:121:29: too many errors"
cmd\\hello\\main.go:35:10: Error return value of `w.Write` is not checked (errcheck)
    w.Write([]byte(err.Error()))
    ^
cmd\\hello\\main.go:39:9: Error return value of `w.Write` is not checked (errcheck)
    w.Write([]byte(msg))
    ^
cmd\\hello\\main.go:51:11: Error return value of `w.Write` is not checked (errcheck)
    w.Write([]byte(err.Error()))
    ^
make: *** [Makefile:4: lint] Error 2
```

Рис.6 — Проверка линтером, нету обработки ошибок в файле с примером.

Вывод: Я научился организовывать долгосрочное хранение данных с помощью PostgreSQL и Golang.