МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра вычислительной техники



**Лабораторная работа №5**

по дисциплине: «WEB - ПРОГРАММИРОВАНИЕ»

на тему: **«Работа с графикой. Построение диаграмм.»**

Вариант №6

Выполнил: Проверил:

Студент гр. АВТ-219: Гунько А. В.

Завёрткин М. А.

Новосибирск

2025

СОДЕРЖАНИЕ

Введение 3

Ход работы 3

Вывод 6

# **Задание**

Задания различаются структурой базы данных, регистрационными записями и паролями пользователей, имеющих разные права доступа к ней, реализованными в соответствии с заданиями к лабораторной работе №1, и являются модификацией лабораторной работы №3.

**SQL запрос на агрегацию данных**

SELECT clients.name, SUM(orders.price) AS sum

FROM clients

LEFT JOIN orders ON client\_id = clients.id

GROUP BY clients.name

**Веб-сервер на GO**

File main.go:

package main

import (

"fmt"

"log"

"net/http"

"slices"

"strconv"

"strings"

)

func index\_handler(w http.ResponseWriter, r \*http.Request) {

request\_id := r.URL.Query().Get("id")

id, err := strconv.Atoi(request\_id)

if err != nil {

id = -1

}

query, orders, err := GetOrders(id)

if err != nil {

http.Error(w, "Internal DB error", http.StatusInternalServerError)

panic(err)

}

var session\_ptr \*SessionEntry

session, err := GetSession(r)

if err == nil {

session\_ptr = &session

} else {

session\_ptr = nil

}

PrintIndex(w, query, orders, session\_ptr)

}

func insert\_handler(w http.ResponseWriter, r \*http.Request) {

if !IsAuthorized(r, USER) {

http.Error(w, "Unauthorized", http.StatusUnauthorized)

return

}

if r.Method == http.MethodGet {

employes := GetIdName("employes")

items := GetIdName("items")

clients := GetIdName("clients")

orderers := GetIdName("orderers")

PrintInsert(w, employes, items, clients, orderers)

}

if r.Method == http.MethodPost {

var err error

employee, err := strconv.Atoi(r.FormValue("employee\_id"))

if err != nil {

http.Error(w, "Invalid employee id", http.StatusBadRequest)

return

}

price, err := strconv.Atoi(r.FormValue("price"))

if err != nil {

http.Error(w, "Invalid price", http.StatusBadRequest)

return

}

item, err := strconv.Atoi(r.FormValue("item\_id"))

if err != nil {

http.Error(w, "Invalid item id", http.StatusBadRequest)

return

}

client, err := strconv.Atoi(r.FormValue("client\_id"))

if err != nil {

http.Error(w, "Invalid client id", http.StatusBadRequest)

return

}

orderer, err := strconv.Atoi(r.FormValue("orderer\_id"))

if err != nil {

http.Error(w, "Invalid orderer id", http.StatusBadRequest)

return

}

err = InsertWithIds(employee, price, item, client, orderer)

if err != nil {

http.Error(w, "Error while inserting", http.StatusBadRequest)

return

}

http.Redirect(w, r, "/index", http.StatusSeeOther)

}

}

func update\_handler(w http.ResponseWriter, r \*http.Request) {

if !IsAuthorized(r, USER) {

http.Error(w, "Unauthorized", http.StatusUnauthorized)

return

}

if r.Method == http.MethodGet {

request\_id := r.URL.Query().Get("id")

id, err := strconv.Atoi(request\_id)

if err != nil {

http.Error(w, "Id not present", http.StatusBadRequest)

return

}

to\_update, err := GetOrderRaw(id)

employes := getIdNameOrdered("employes", to\_update.EmployeeId)

items := getIdNameOrdered("items", to\_update.ItemId)

clients := getIdNameOrdered("clients", to\_update.ClientId)

orderers := getIdNameOrdered("orderers", to\_update.OrdererId)

PrintUpdate(w, id, employes, to\_update.Price, items, clients, orderers)

}

if r.Method == http.MethodPost {

var err error

id, err := strconv.Atoi(r.FormValue("id"))

if err != nil {

http.Error(w, "Invalid order id", http.StatusBadRequest)

return

}

employee, err := strconv.Atoi(r.FormValue("employee\_id"))

if err != nil {

http.Error(w, "Invalid employee id", http.StatusBadRequest)

return

}

price, err := strconv.Atoi(r.FormValue("price"))

if err != nil {

http.Error(w, "Invalid price", http.StatusBadRequest)

return

}

item, err := strconv.Atoi(r.FormValue("item\_id"))

if err != nil {

http.Error(w, "Invalid item id", http.StatusBadRequest)

return

}

client, err := strconv.Atoi(r.FormValue("client\_id"))

if err != nil {

http.Error(w, "Invalid client id", http.StatusBadRequest)

return

}

orderer, err := strconv.Atoi(r.FormValue("orderer\_id"))

if err != nil {

http.Error(w, "Invalid orderer id", http.StatusBadRequest)

return

}

err = UpdateWithIds(id, employee, price, item, client, orderer)

if err != nil {

http.Error(w, "Error while updating", http.StatusBadRequest)

return

}

http.Redirect(w, r, "/index", http.StatusSeeOther)

}

}

func getIdNameOrdered(table string, id int) []IdName {

result := GetIdName(table)

for i, idName := range result {

if idName.Id == id {

tmp := result[i]

result = append(result[:i], result[i+1:]...)

result = slices.Insert(result, 0, tmp)

}

}

return result

}

func delete\_handler(w http.ResponseWriter, r \*http.Request) {

if !IsAuthorized(r, ADMIN) {

http.Error(w, "Unauthorized", http.StatusUnauthorized)

return

}

request\_id := r.URL.Query().Get("id")

id, err := strconv.Atoi(request\_id)

if err != nil {

http.Error(w, "Invalid id", http.StatusBadRequest)

return

}

err = Delete(id)

if err != nil {

http.Error(w, "Wrong id", http.StatusBadRequest)

return

}

http.Redirect(w, r, "/index", http.StatusSeeOther)

}

var login\_error\_str = "Invalid login"

func login\_handler(w http.ResponseWriter, r \*http.Request) {

if r.Method == http.MethodGet {

PrintLogin(w, nil)

}

if r.Method == http.MethodPost {

username := r.FormValue("username")

password := r.FormValue("password")

user, err := GetUserRecord(username)

if err == nil && strings.Compare(password, user.Password) == 0 {

err = CreateSession(w, r, user)

}

if err != nil {

PrintLogin(w, &login\_error\_str)

} else {

http.Redirect(w, r, "/index", http.StatusSeeOther)

}

}

}

func logout\_handler(w http.ResponseWriter, r \*http.Request) {

DeleteCurrentSession(w, r)

http.Redirect(w, r, "/index", http.StatusSeeOther)

}

func histogram\_handler(w http.ResponseWriter, r \*http.Request) {

query, orders, err := GetAggregatedData()

if err != nil {

http.Error(w, "Internal DB error", http.StatusInternalServerError)

panic(err)

}

PrintHistogram(w, query, orders)

}

func main() {

var err error

err = InitModel()

if err != nil {

log.Fatal(err)

return

}

defer DeinitModel()

http.HandleFunc("/index", index\_handler)

http.HandleFunc("/insert", insert\_handler)

http.HandleFunc("/update", update\_handler)

http.HandleFunc("/delete", delete\_handler)

http.HandleFunc("/login", login\_handler)

http.HandleFunc("/logout", logout\_handler)

http.HandleFunc("/histogram", histogram\_handler)

http.Handle("/static/", http.StripPrefix("/static/", http.FileServer(http.Dir("static"))))

fmt.Println("Start listening at http://localhost:8080/index")

log.Fatal(http.ListenAndServe(":8080", nil))

}

File model.go:

package main

import (

"database/sql"

"errors"

"fmt"

\_ "github.com/lib/pq"

"log"

"strings"

)

var db \*sql.DB

func InitModel() error {

var err error

db, err = sql.Open("postgres", "user=postgres dbname=orders sslmode=disable")

if err != nil {

log.Fatal(err)

return err

}

return nil

}

func DeinitModel() {

db.Close()

}

type Order struct {

Id int

Employee string

Price int

Item string

Client string

Orderer string

}

type OrderRaw struct {

Id int

EmployeeId int

Price int

ItemId int

ClientId int

OrdererId int

}

const sql\_select = `

SELECT orders.id,employes.name,orders.price,items.name,clients.name,orderers.name

FROM orders

LEFT JOIN employes ON employee\_id = employes.id

LEFT JOIN items ON item\_id = items.id

LEFT JOIN clients ON client\_id = clients.id

LEFT JOIN orderers ON orderer\_id = orderers.id

`

func GetOrders(id int) (string, []Order, error) {

result := make([]Order, 0, 8)

cur\_select := sql\_select

if id >= 0 {

cur\_select += fmt.Sprintf("WHERE orders.id = %d", id)

} else {

cur\_select += fmt.Sprintf("ORDER BY orders.id")

}

rows, err := db.Query(cur\_select)

if err != nil {

return "", nil, err

}

defer rows.Close()

for rows.Next() {

var order Order

err := rows.Scan(&order.Id, &order.Employee, &order.Price, &order.Item, &order.Client, &order.Orderer)

if err != nil {

fmt.Println(err)

continue

}

result = append(result, order)

}

return cur\_select, result, nil

}

const aggregated\_data\_query = `

SELECT clients.name, SUM(orders.price) AS sum

FROM clients

LEFT JOIN orders ON client\_id = clients.id

GROUP BY clients.name

`

type ClientAndSum struct {

Client string

Sum int

}

func GetAggregatedData() (string, []ClientAndSum, error) {

result := make([]ClientAndSum, 0, 8)

rows, err := db.Query(aggregated\_data\_query)

if err != nil {

return "", nil, err

}

defer rows.Close()

for rows.Next() {

var result\_element ClientAndSum

err := rows.Scan(&result\_element.Client, &result\_element.Sum)

if err != nil {

fmt.Println(err)

continue

}

result = append(result, result\_element)

}

return aggregated\_data\_query, result, nil

}

func GetOrderRaw(id int) (OrderRaw, error) {

var result OrderRaw

cur\_select := `

SELECT id,employee\_id,price,item\_id,client\_id,orderer\_id

FROM orders

WHERE id = %d

`

err := db.QueryRow(fmt.Sprintf(cur\_select, id)).Scan(&result.Id, &result.EmployeeId, &result.Price, &result.ItemId, &result.ClientId, &result.OrdererId)

if err != nil {

return result, err

}

return result, nil

}

type IdName struct {

Id int

Name string

}

func GetIdName(table string) []IdName {

result := make([]IdName, 0, 8)

rows, err := db.Query(fmt.Sprintf("SELECT id,name FROM %s", table))

if err != nil {

panic(err)

}

defer rows.Close()

for rows.Next() {

var tmp IdName

err := rows.Scan(&tmp.Id, &tmp.Name)

if err != nil {

fmt.Println(err)

continue

}

result = append(result, tmp)

}

return result

}

func Delete(id int) error {

\_, err := db.Exec("DELETE FROM orders WHERE id = $1", id)

return err

}

const sql\_insert\_tmpl = `

INSERT INTO orders (employee\_id, price, item\_id, client\_id, orderer\_id) VALUES

($1, $2, $3, $4, $5);

`

func InsertWithIds(employee int, price int, item int, client int, orderer int) error {

\_, err := db.Exec(sql\_insert\_tmpl, employee, price, item, client, orderer)

return err

}

const sql\_update\_tmpl = `

UPDATE orders

SET employee\_id=$1, price=$2, item\_id=$3, client\_id=$4, orderer\_id=$5

WHERE id = $6

`

func UpdateWithIds(id int, employee int, price int, item int, client int, orderer int) error {

\_, err := db.Exec(sql\_update\_tmpl, employee, price, item, client, orderer, id)

return err

}

type UserRole int

const (

USER UserRole = 0

ADMIN UserRole = 1

)

type UserRecord struct {

Username string

Password string

Role UserRole

}

const sql\_select\_user = `

SELECT username,password,role

FROM users

WHERE username = '%s'

`

func GetUserRecord(username string) (UserRecord, error) {

var result UserRecord

var role\_str string

err := db.QueryRow(fmt.Sprintf(sql\_select\_user, username)).Scan(&result.Username, &result.Password, &role\_str)

if err != nil {

return result, err

}

if strings.Compare(role\_str, "user") == 0 {

result.Role = USER

} else if strings.Compare(role\_str, "admin") == 0 {

result.Role = ADMIN

} else {

return result, errors.New("Invalid user role in DB")

}

return result, nil

}

File sessions.go:

package main

import (

"errors"

"github.com/gorilla/sessions"

"net/http"

)

const session\_name = "session-name"

var store = sessions.NewCookieStore([]byte("secret-key"))

type SessionEntry struct {

User UserRecord

}

func GetSession(r \*http.Request) (SessionEntry, error) {

var result SessionEntry

session, \_ := store.Get(r, session\_name)

if session.IsNew {

return result, errors.New("Session doesn't exist")

}

var ok bool

result.User.Username, ok = session.Values["username"].(string)

if !ok {

return result, errors.New("Username for session not set")

}

var user\_role int

user\_role, ok = session.Values["role"].(int)

if !ok {

return result, errors.New("User role for session not set")

}

result.User.Role = UserRole(user\_role)

return result, nil

}

func IsAuthorized(r \*http.Request, role UserRole) bool {

session, err := GetSession(r)

if err != nil {

return false

}

return session.User.Role >= role

}

func CreateSession(w http.ResponseWriter, r \*http.Request, user UserRecord) error {

var user\_role int

user\_role = int(user.Role)

session := sessions.NewSession(store, session\_name)

session.Values["username"] = user.Username

session.Values["role"] = user\_role

err := session.Save(r, w)

return err

}

func DeleteCurrentSession(w http.ResponseWriter, r \*http.Request) error {

session, \_ := store.Get(r, session\_name)

if session.IsNew {

return errors.New("Session doesn't exist")

}

session.Options.MaxAge = -1

err := session.Save(r, w)

return err

}

File view.go:

package main

import (

"fmt"

"io"

"net/http"

"strings"

)

const search\_form\_html = `

<form method="GET" action="">

<p>ID:<input type="number" step="1" min="1" name="id">

<input type="SUBMIT" value="Search"></p></form>

<a href="/index">Show all</a>

<a href="/histogram">View histogram</a>

`

const table\_end = `

</tbody></table>

`

func make\_table\_header(w io.Writer, role \*UserRole) {

fmt.Fprintf(w, "%s", `

<table border="1"><tbody>

<tr><th><b>id</b></th><th><b>employee</b></th><th><b>price</b></th><th><b>item</b></th><th><b>client</b></th><th><b>orderer</b></th>`)

if role != nil {

fmt.Fprintf(w, "%s", `<th><b>Update</b></th>`)

if \*role >= ADMIN {

fmt.Fprintf(w, "%s", `<th><b>Delete</b></th>`)

}

}

fmt.Fprintf(w, "%s", `</tr>`)

}

func make\_row(w io.Writer, id int, employee string, price int, item string, client string, orderer string, role \*UserRole) {

fmt.Fprintf(w, `<tr><td>%d</td><td>%s</td><td>%d</td><td>%s</td><td>%s</td><td>%s</td>`, id, employee, price, item, client, orderer)

if role != nil {

fmt.Fprintf(w, `<td><a href="update?id=%d">Update</a></td>`, id)

if \*role >= ADMIN {

fmt.Fprintf(w, `<td><a href="delete?id=%d">Delete</a></td>`, id)

}

}

fmt.Fprintf(w, `</tr>`)

}

func PrintIndex(w http.ResponseWriter, query string, orders []Order, session \*SessionEntry) {

fmt.Fprintf(w, "<h1>Just a db, bro</h1><div>")

fmt.Fprintf(w, "<pre>%s</pre>", query)

var role \*UserRole = nil

if session != nil {

role = &session.User.Role

}

make\_table\_header(w, role)

for \_, order := range orders {

make\_row(w, order.Id, order.Employee, order.Price, order.Item, order.Client, order.Orderer, role)

}

fmt.Fprintf(w, "%s", table\_end)

fmt.Fprintf(w, search\_form\_html)

fmt.Fprintf(w, `<br/>`)

if session != nil {

fmt.Fprintf(w, `<a href="/insert">Insert</a>`)

fmt.Fprintf(w, `<div>Logged as %s</div>`, session.User.Username)

fmt.Fprintf(w, `<a href="/logout">Log out</a>`)

} else {

fmt.Fprintf(w, `<a href="/login">Login</a>`)

}

fmt.Fprintf(w, "</div>")

}

func PrintInsert(w http.ResponseWriter, employes []IdName, items []IdName, clients []IdName, orderers []IdName) {

fmt.Fprintf(w, "<html><form method=\"POST\" action=\"/insert\"><p>")

fmt.Fprintf(w, "Employee<select name=\"employee\_id\">")

print\_options(w, employes)

fmt.Fprintf(w, "</select>")

fmt.Fprintf(w, "Price<input type=\"number\" step=\"1\" min=\"1\" name=\"price\"/>")

fmt.Fprintf(w, "Item<select name=\"item\_id\">")

print\_options(w, items)

fmt.Fprintf(w, "</select>")

fmt.Fprintf(w, "Client<select name=\"client\_id\">")

print\_options(w, clients)

fmt.Fprintf(w, "</select>")

fmt.Fprintf(w, "Orderer<select name=\"orderer\_id\">")

print\_options(w, orderers)

fmt.Fprintf(w, "</select>")

fmt.Fprintf(w, "<input type=\"SUBMIT\" value=\"Add\"></p></form></html>")

}

func PrintUpdate(w http.ResponseWriter, id int, employes []IdName, price int, items []IdName, clients []IdName, orderers []IdName) {

fmt.Fprintf(w, "<html><form method=\"POST\" action=\"/update\"><p>")

fmt.Fprintf(w, "Employee<select name=\"employee\_id\">")

print\_options(w, employes)

fmt.Fprintf(w, "</select>")

fmt.Fprintf(w, "Price<input type=\"number\" step=\"1\" min=\"1\" name=\"price\" value=\"%d\"/>", price)

fmt.Fprintf(w, "Item<select name=\"item\_id\">")

print\_options(w, items)

fmt.Fprintf(w, "</select>")

fmt.Fprintf(w, "Client<select name=\"client\_id\">")

print\_options(w, clients)

fmt.Fprintf(w, "</select>")

fmt.Fprintf(w, "Orderer<select name=\"orderer\_id\">")

print\_options(w, orderers)

fmt.Fprintf(w, "</select>")

fmt.Fprintf(w, "<input type=\"hidden\" name=\"id\" value=\"%d\"/>", id)

fmt.Fprintf(w, "<input type=\"SUBMIT\" value=\"Update\"></p></form></html>")

}

func PrintLogin(w http.ResponseWriter, error \*string) {

fmt.Fprintf(w, "<html><form method=\"POST\" action=\"/login\"><p>")

fmt.Fprintf(w, "Login<input name=\"username\"/>")

fmt.Fprintf(w, "Password<input name=\"password\" type=\"password\"/>")

fmt.Fprintf(w, "<input type=\"SUBMIT\" value=\"Login\"></p></form>")

if error != nil {

fmt.Fprintf(w, "Error: %s", \*error)

}

fmt.Fprintf(w, "</html>")

}

func print\_options(w io.Writer, table []IdName) {

for \_, tmp := range table {

fmt.Fprintf(w, "<option value=\"%d\">%s</option>", tmp.Id, tmp.Name)

}

}

const histogram\_script = `

<script src="static/chart.js"></script>

<script>

const ctx = document.getElementById('histogram');

new Chart(ctx, {

type: 'bar',

data: {

labels: [%s],

datasets: [{

data: [%s],

borderWidth: 1

}]

},

options: {

scales: {

y: {

beginAtZero: true

}

}

}

});

</script>

`

func PrintHistogram(w http.ResponseWriter, query string, data []ClientAndSum) {

fmt.Fprintf(w, "<h1>Orders sum by clients</h1><div>")

fmt.Fprintf(w, "<pre>%s</pre>", query)

var labels strings.Builder

var values strings.Builder

for \_, element := range data {

fmt.Fprintf(&labels, "'%s',", element.Client)

fmt.Fprintf(&values, "%d,", element.Sum)

}

fmt.Fprintf(w, `<div><canvas id="histogram"/></div>`)

fmt.Fprintf(w, histogram\_script, labels.String(), values.String())

fmt.Fprintf(w, "</div>")

}