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 «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 6

по курсу «Компьютерные системы и сети»

«Импорт новостей в базу данных из RSS-канала»

Студент группы ИУ9-32Б Федуков А. А.

Преподаватель Посевин Д. П.

5 ноября 2024 г.

Цель работы

Целью данной работы является разработка приложения выполняющего разбор RSS-ленты новостей (по вариантам) и запись новостей в таблицу базы данных MySQL.

Задание

Приложение должно подключаться к удаленной базе данных под управлением СУБД MySQL и выполнять обновление новостей в таблице. При этом важно учесть следующее: при повторном запуске приложения необходимо сравнивать новости в rss канале с теми, что записаны в таблице и при повторных запусках приложения дублей новостей в таблице быть не должно. Вывод обновления данных должен происходить асинхронно в html dashboard. Во время проверки работы будет проверяться следующее: выполняется обновление таблицы из rss канала, далее удаляем несколько записей из таблицы в базе данных и еще раз запускаем приложение; удаленные записи должны появиться в таблице. Также возможно и другой метод тестирования: берем запись какой-то новости в таблице, в текст новости или заголовок новости вносим произвольные строки, запускаем приложение и проверяем добавилась ли новость, текст которой мы испортили из rss канала в таблицу или нет — должна добавляться.

Реализация

Я создал парсер RSS, который загружался в базу данных и отправлял свой статус на дашборд.

Код приложения

Листинг 1: Файл main.go

```
1 package main
2
3 import (
4   "os"
5   "time"
```

```
)

var MyIP = "127.0.0.1"
var MyPort = ":4411"
var RSS_Source = "https://news.rambler.ru/rss/Magadan/"

type Info struct {
  Id     int    `json:"id"`
  Header string `json:"header"`
  Text   string `json:"text"`
}

func main() {
  MyIP = os.Getenv("MyIP")

  go startSocket()

  for {
    infos := parseRSS()

    sendClear()
    time.Sleep(10 * time.Second)
    getFromEstonia(&infos)
    for _, v := range infos {
      sendToSocket(v)
    }

    infos = parseRSS()
    // loadToEstonia(&infos)
    loadToEstonia(&infos)

    time.Sleep(10 * time.Second)
  }
}
```

Листинг 2: Файл parseRSS.go

```
package main

import (
  "log"
  "strconv"

  "github.com/SlyMarbo/rss"
)

func parseRSS() []Info {
```

```
11    log.Println("Parse RSS")
12
13    rssObject, err := rss.Fetch(RSS_Source)
14    infos := make([]Info, 0)
15    if err == nil {
16      // s += fmt.Sprintf("<h3>Title:</h3><i>%s\n", rssObject.Title)
17      // s += fmt.Sprintf("</i><b>Description:</b> </br><i>%s\n",
      rssObject.Description)
18      // s += fmt.Sprintf("</br></i><b>Number of Items:</b> </br><i>%d\n",
      len(rssObject.Items))
19      // s += "</br>"
20      for _, v := range rssObject.Items {
21        id, err := strconv.Atoi(v.ID)
22        if err != nil {
23          log.Println("Error with id")
24        } else {
25          infos = append(infos, Info{Header: v.Title, Text: v.Summary, Id:
      id})
26        }
27      }
28    } else {
29      log.Println("Error with rss")
30    }
31
32    return infos
33  }
```

Листинг 3: Файл sendToSocket.go

```
1  package main
2
3  import (
4    "encoding/json"
5    "log"
6    "net/http"
7    "sync"
8
9    "github.com/gorilla/websocket"
10 )
11
12 // Global variables to hold connected clients and mutex to avoid race
       conditions
13 var clients = make(map[*websocket.Conn]bool)
14 var mu sync.Mutex
15
16 // Upgrader to upgrade HTTP connections to WebSocket connections
17 var upgrader = websocket.Upgrader{
```

```
18    CheckOrigin: func(r *http.Request) bool {
19      return true // Allow all connections
20    },
21  }
22
23  // Function to handle incoming WebSocket connections
24  func handleConnection(w http.ResponseWriter, r *http.Request) {
25    // Upgrade the HTTP connection to a WebSocket connection
26    conn, err := upgrader.Upgrade(w, r, nil)
27    if err != nil {
28      log.Println("Error upgrading connection:", err)
29      return
30    }
31    defer conn.Close()
32
33    // Add the new connection to the clients map
34    mu.Lock()
35    clients[conn] = true
36    mu.Unlock()
37    log.Println("New client connected")
38
39    // Listen for messages from this client
40    for {
41      _, _, err := conn.NextReader() // Keep the connection open to listen
        for messages
42      if err != nil {
43        log.Println("Client disconnected:", err)
44        break
45      }
46    }
47  }
48
49  func sendToClientInfo(conn *websocket.Conn, logMsg Info) error {
50    if conn != nil {
51      jsonData, err := json.Marshal(logMsg)
52      if err != nil {
53        log.Println("Error marshaling JSON:", err)
54        return err
55      }
56
57      err = conn.WriteMessage(websocket.TextMessage, jsonData)
58      if err != nil {
59        log.Println("Error sending message:", err)
60        return err
61      }
62      // fmt.Println("Sended data to socket")
```

```
63      return nil
64    }
65    return http.ErrAbortHandler
66  }
67
68  func sendClear()  {
69    mu.Lock()
70    log.Println("Send to socket")
71    defer mu.Unlock()
72    // Loop through all connected clients and send the message
73    for client := range clients {
74      msg := "clear"
75      err := client.WriteMessage(websocket.TextMessage, []byte(msg))
76      if err != nil {
77        log.Println("Error sending message:", err)
78
79      }
80    }
81
82  }
83  func sendToSocket(info Info) {
84    mu.Lock()
85    log.Println("Send to socket")
86    defer mu.Unlock()
87    // Loop through all connected clients and send the message
88    for client := range clients {
89      err := sendToClientInfo(client, info)
90      if err != nil {
91        log.Println("Error sending message:", err)
92        client.Close()
93        delete(clients, client) // Remove client if there's an error
94      }
95    }
96  }
97
98  func startSocket() {
99    // Handle WebSocket connections
100   http.HandleFunc("/ws", handleConnection)
101
102   log.Println("Socket server started on", MyIP+MyPort)
103   // Start the server
104   err := http.ListenAndServe(MyIP+MyPort, nil)
105   if err != nil {
106     log.Fatal("ListenAndServe: ", err)
107   }
108 }
```

Листинг 4: Файл loadToEstonia.go

```
1  package main
2
3  import (
4    "database/sql"
5    "fmt"
6    "log"
7
8    _ "github.com/go-sql-driver/mysql"
9  )
10
11 const (
12   username  = "iu9networkslabs"
13   password  = "Je2dTYr6"
14   hostname  = "students.yss.su"
15   dbName    = "iu9networkslabs"
16   tableName = "fedukov_lab5"
17 )
18
19 // INSERT INTO `fedukov_lab5` (`header`, `text`)
20 // VALUES ('ABOBA', 'Текст абобы');
21
22 // SELECT * FROM `fedukov_lab5` LIMIT 50
23
24 func queryData(db *sql.DB, query string, args ...interface{}) []Info {
25   rows, err := db.Query(query, args...)
26   if err != nil {
27     panic(fmt.Errorf("error executing query: %v", err))
28   }
29
30   defer rows.Close()
31   out := make([]Info, 0)
32
33   for rows.Next() {
34     info := Info{}
35     if err := rows.Scan(&info.Id, &info.Header, &info.Text); err != nil
       {
36       panic(fmt.Errorf("error scanning row: %v", err))
37     }
38     out = append(out, info)
39   }
40
41   if err := rows.Err(); err != nil {
42     panic(fmt.Errorf("rows error: %v", err))
43   }
44
```

```
45    return out
46  }
47
48  func queryOneData(db *sql.DB, query string, args ...interface{}) Info {
49    rows, err := db.Query(query, args...)
50    if err != nil {
51      panic(fmt.Errorf("error executing query: %v", err))
52    }
53
54    defer rows.Close()
55    out := make([]Info, 0)
56
57    for rows.Next() {
58      info := Info{}
59      if err := rows.Scan(&info.Id, &info.Header, &info.Text); err != nil
        {
60        panic(fmt.Errorf("error scanning row: %v", err))
61      }
62      out = append(out, info)
63    }
64
65    if err := rows.Err(); err != nil {
66      panic(fmt.Errorf("rows error: %v", err))
67    }
68
69    if len(out) == 0 {
70      out = append(out, Info{Header: "NO DATA"})
71    }
72
73    return out[0]
74
75  }
76
77  func insertData(db *sql.DB, info *Info) {
78    result, err := db.Exec(fmt.Sprintf("INSERT INTO %s (id, header, text)
       VALUES (?, ?, ?)", tableName), info.Id, info.Header, info.Text)
79    if err != nil {
80      panic(fmt.Errorf("error inserting data: %v", err))
81    }
82
83    rowsAffected, err := result.LastInsertId()
84    if err != nil {
85      panic(fmt.Errorf("error getting last insert id: %v", err))
86    }
87    fmt.Printf("Inserted: %+v (%d times)\n", info, rowsAffected)
88
```

```
89 }
90
91 func updateData(db *sql.DB, info *Info) {
92   result, err := db.Exec(fmt.Sprintf("UPDATE %s SET header = ?, text = ?
       WHERE id = ?", tableName), info.Header, info.Text, info.Id)
93   if err != nil {
94     panic(fmt.Errorf("error updating data: %v", err))
95   }
96
97   rowsAffected, err := result.RowsAffected()
98   if err != nil {
99     panic(fmt.Errorf("error getting affected rows: %v", err))
100  }
101  fmt.Printf("Updated: %+v (%d times)\n", info, rowsAffected)
102
103 }
104
105 func deleteData(db *sql.DB, info *Info) {
106   result, err := db.Exec(fmt.Sprintf("DELETE FROM %s WHERE id = ?",
       tableName), info.Id)
107   if err != nil {
108     panic(fmt.Errorf("error deleting data: %v", err))
109   }
110
111   rowsAffected, err := result.RowsAffected()
112   if err != nil {
113     panic(fmt.Errorf("error getting affected rows: %v", err))
114   }
115   fmt.Printf("Deleted: %+v (%d times)\n", info, rowsAffected)
116 }
117
118 func fitInTable(db *sql.DB, info *Info) {
119   oldInfo := queryData(db, fmt.Sprintf("select * from %s WHERE id = ?",
       tableName), info.Id)
120
121   switch len(oldInfo) {
122   case 0:
123     insertData(db, info)
124   case 1:
125     updateData(db, info)
126   default:
127     deleteData(db, info)
128     insertData(db, info)
129   }
130 }
131
```

```
132 func loadToEstonia(infos *[]Info) {
133   log.Println("Process database")
134
135   db, err := sql.Open("mysql", fmt.Sprintf("%s:%s@tcp(%s)/%s", username,
        password, hostname, dbName))
136   if err != nil {
137     panic(err.Error())
138   }
139
140   // Verify the connection
141   if err := db.Ping(); err != nil {
142     panic(fmt.Errorf("error connecting to database: %v", err))
143   }
144
145   for _, v := range *infos {
146     fitInTable(db, &v)
147   }
148
149   // updateData(db, Info{1, "Alice", "ssasassas"})
150   // deleteData(db, Info{Id: 2})
151   // out := queryData(db, fmt.Sprintf("select * from %s WHERE id = ?",
       tableName), 1)
152   // fmt.Println(out)
153
154   defer db.Close()
155   // fmt.Println("Success!")
156 }
157
158 func getFromEstonia(infos *[]Info) {
159   log.Println("Process database")
160
161   db, err := sql.Open("mysql", fmt.Sprintf("%s:%s@tcp(%s)/%s", username,
        password, hostname, dbName))
162   if err != nil {
163     panic(err.Error())
164   }
165
166   // Verify the connection
167   if err := db.Ping(); err != nil {
168     panic(fmt.Errorf("error connecting to database: %v", err))
169   }
170
171   // for i, _ := range *infos {
172   //   // fitInTable(db, &v)
173   //   (*infos)[i] = queryData(db, fmt.Sprintf("select * from %s",
       tableName))
```

```
174     // }
175     *infos = queryData(db, fmt.Sprintf("select * from %s", tableName))
176     // *infos = queryData(db, fmt.Sprintf("select * from %s WHERE id = ?",
            tableName), info.Id)
177     // updateData(db, Info{1, "Alice", "ssasassas"})
178     // deleteData(db, Info{Id: 2})
179     // out := queryData(db, fmt.Sprintf("select * from %s WHERE id = ?",
            tableName), 1)
180     // fmt.Println(out)
181
182     defer db.Close()
183     // fmt.Println("Success!")
184 }
```

Листинг 5: Файл start.sh

```
1  #!/bin/bash
2
3  MyIP=$(hostname -I | awk '{ print $1 }' | tr -d '[:space:]')
4  export MyIP
5
6  SCRIPT_DIR=$( cd -- "$( dirname -- "${BASH_SOURCE[0]}" )" &> /dev/null
       && pwd ) # Lab files dir. Default is script's dir
7  cd "$SCRIPT_DIR" || exit 1
8
9  echo "Building..."
10 cd main || exit 1
11 rm go.mod go.sum
12 go mod init main
13 # Import
14 # go get "github.com/SlyMarbo/rss"
15 # go get "github.com/gorilla/websocket"
16 # go get github.com/go-sql-driver/mysql
17 go mod tidy
18
19 go build -o ../fedukov_lab5
20 killall fedukov_lab5 2>/dev/null
21 echo "Running..."
22 ../fedukov_lab5
```

Код дашборда

Листинг 6: Файл index.html

```
1  <html lang="en">
```

```
2
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0
      ">
6     <link rel="stylesheet" href="style.css">
7     <title>MySQL dash</title>
8 </head>
9
10 <body>
11     <div id="log">
12         Connection status
13     </div>
14     <h1 id="newsHeader">News: 0</h1>
15     <div class="news">
16         <template id="InfoTemplate">
17             <div class="new">
18                 <h2 class="header"></h2>
19                 <p class="text"></p>
20                 <hr>
21             </div>
22         </template>
23     </div>
24     <script src="socket.js"></script>
25 </body>
26
27 </html>
```

Листинг 7: Файл socket.js

```
1 var INFOs = [];
2 var log = document.getElementById("log");
3 var newsHeader = document.getElementById("newsHeader");
4 var NEWs = document.getElementsByClassName("news")[0];
5 var temp = document.getElementById("InfoTemplate")
6
7 /**
8  *
9  * @param {Node} node
10  * @param {{header : "", text : "", id : 0}} data
11  */
12 function changeNode(node, data) {
13   node.querySelector('.header').innerHTML = data.Number+1 + ". " + data.
       header
14   node.querySelector('.text').textContent = data.text
15   return node
16 }
```

```
17
18 function clearNews(){
19   document.getElementsByClassName("new").length
20   l = document.getElementsByClassName("new").length
21   for (let i = 0; i < l; i++) {
22     document.getElementsByClassName("new")[0].remove()
23   }
24   INFOs = []
25 }
26 function handleInfo(data) {
27   const newInfo = JSON.parse(data)
28
29   // Update existing infos
30   for (let i = 0; i < INFOs.length; i++) {
31     const info = INFOs[i];
32     if (info.id == newInfo.id && document.getElementById(newInfo.id) !=
       null) {
33       newInfo.Number = i
34       changeNode(document.getElementById(newInfo.id), newInfo)
35       return
36     }
37
38   }
39
40
41   let clon = temp.content.cloneNode(true);
42   const newDiv = clon.querySelector('.new');
43   newDiv.id = newInfo.id
44   newInfo.Number = INFOs.length
45   NEWs.appendChild(changeNode(newDiv, newInfo))
46   INFOs.push(document.getElementById(newDiv.id))
47
48 }
49
50 function reconnect () {
51   // Wait for 2 seconds and reconnect
52   setTimeout(() => {
53     initSocket()
54   }, 2000);
55 }
56
57 function initSocket(){
58   var ip = "185.102.139.168"
59   var port = "4411"
60   var socket = new WebSocket("ws://" + ip + ":" + port + "/ws")
61
```

```
62    socket.onopen = function() {
63        log.textContent = "\nСоединение установлено.";
64      };
65
66    socket.onclose = function(event) {
67        if (event.wasClean) {
68          console.log('Соединение закрыто чисто');
69        } else {
70          console.log('Обрыв соединения'); // например, "убит" процесс сер
      вера
71        }
72        console.log('Код: ' + event.code + ' причина: ' + event.reason);
73        // socket.close()
74        log.textContent = "\nСоединение закрыто.";
75        reconnect()
76      };
77
78    socket.onmessage = function(event) {
79        console.log("Получены данные " + event.data);
80        if (event.data == "clear") {
81          clearNews()
82        } else {
83          handleInfo(event.data)
84          newsHeader.textContent = "News: " + INFOs.length
85
86        }
87
88
89      };
90
91    socket.onerror = function(error) {
92        console.log("Ошибка " + error.message);
93        log.textContent = "\nОшибка соединения";
94        // socket.close()
95        // reconnect()
96      };
97  }
98
99  window.onload = initSocket
```

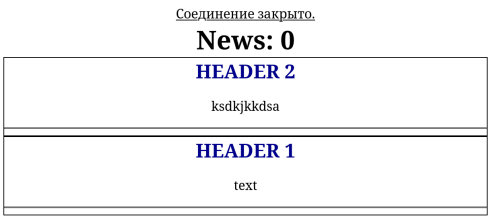
Листинг 8: Файл style.css

```
1  body {
2      margin: 0;
3      padding: 0;
4      text-align: center;
5  }
```

```
6
7  #log {
8      text-decoration: underline;
9  }
10 .news {
11     text-align: center;
12     display: flex;
13     flex-direction: column-reverse;
14     align-items: center;
15     justify-content: flex-start;
16     width: 50%;
17     margin-left: auto;
18     margin-right: auto;
19
20 }
21
22 #newsHeader {
23     margin: 0;
24 }
25
26 .new {
27     width: 75%;
28     border: 1px solid black;
29 }
30
31 .header {
32     color: darkblue;
33     margin: 0;
34 }
35
36 .text {
37
38 }
```

Вывод программы

Программа каждые 10 секунд обновляла новости в базе, после чего отправляла на дашборд.



Вывод

В этот лабораторной я научился подключаться к базе данных используя Go, кроме того я вспомнил особенности RSS и WebSocket.