Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОННИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №2

по дисциплине «Модели данных и системы управления базами данных»

по теме «Интернет площадка услуг по аренде легковых автомобилией(каршеринг)»

Выполнил:

студент гр. 753502

Горбачев Д.А.

Проверил:

Алексеев И.Г.

Минск 2020

* Авторизация

SELECT \* FROM `client` WHERE `login` = @uL AND `password` = @uP AND `status\_id` = 0;

* Управление пользователями

Создание:

LOCK TABLES client WRITE;

INSERT INTO client (email, login, password, phonenumber, passport\_id, adress\_id, role\_id, status\_id) VALUES (@uEmail, @uLogin, @uPassword, @uPhone, @uPassport, @uAdress, @uRole, @uStatus);

UNLOCK TABLES;

Удаление:

LOCK TABLES client WRITE;

SELECT \* FROM `client` WHERE `login` = @uL;

DELETE FROM client WHERE `login` = @uL;

UNLOCK TABLES;

Изменение прав:

UPDATE client SET `role\_id` = @role WHERE `login` = @login;

Блокировка:

LOCK TABLES client WRITE;

UPDATE client SET `status\_id` = @uS WHERE `login` = @uL;

UNLOCK TABLES;

Блокировка на определенное время:

LOCK TABLES client WRITE;

CREATE EVENT IF NOT EXISTS event

ON SCHEDULE AT CURRENT\_TIMESTAMP + INTERVAL @time MINUTE

DO UPDATE client SET `status\_id` = 0 WHERE `login` = @uL";

UNLOCK TABLES;

Поиск пользователей:

SELECT

`client`.id, phoneNumber, email, register\_date, patrynomic, firstname, secondname , status.name as status, role.name as role

FROM `client`

INNER JOIN `passport` ON `client`.passport\_id = `passport`.id

Inner join patrynomic on `passport`.patrynomic\_id = `patrynomic`.id

inner join firstname on `passport`.patrynomic\_id = `firstname`.id

inner join secondname on `passport`.patrynomic\_id = `secondname`.id

inner join status on `client`.status\_id = `status`.id

inner join role on `client`.role\_id = `role`.id

WHERE

`login` = … AND `role`.name = … AND `status`.name = … AND `firstname` =… AND `secondname` = … `patrynomic` = AND `email` =AND `phoneNumber` = …

* Журналирование

CREATE TABLE IF NOT EXISTS `app\_schema`.`insert\_log` (

`id` MEDIUMINT NOT NULL AUTO\_INCREMENT,

`user\_id` MEDIUMINT NOT NULL,

`email` VARCHAR(45) NULL,

`login` VARCHAR(45) NOT NULL,

`password` INT(8) NOT NULL,

`phoneNumber` VARCHAR(45) NULL,

`passport\_id` INT NOT NULL,

`adress\_id` MEDIUMINT NOT NULL,

`role\_id` TINYINT NOT NULL,

`status\_id` TINYINT NOT NULL,

`register\_date` DATETIME NOT NULL,

`time` TIMESTAMP NOT NULL,

PRIMARY KEY (`id`))

CREATE TABLE IF NOT EXISTS `app\_schema`.`delete\_log` (

`id` MEDIUMINT NOT NULL AUTO\_INCREMENT,

`user\_id` MEDIUMINT NOT NULL,

`email` VARCHAR(45) NULL,

`login` VARCHAR(45) NOT NULL,

`password` INT(8) NOT NULL,

`phoneNumber` VARCHAR(45) NULL,

`passport\_id` INT NOT NULL,

`adress\_id` MEDIUMINT NOT NULL,

`role\_id` TINYINT NOT NULL,

`status\_id` TINYINT NOT NULL,

`register\_date` DATETIME NOT NULL,

`time` TIMESTAMP NOT NULL,

PRIMARY KEY (`id`))

CREATE TABLE IF NOT EXISTS `app\_schema`.`update\_log` (

`id` MEDIUMINT NOT NULL AUTO\_INCREMENT,

`user\_id` MEDIUMINT NOT NULL,

`email` VARCHAR(45) NULL,

`login` VARCHAR(45) NOT NULL,

`password` INT(8) NOT NULL,

`phoneNumber` VARCHAR(45) NULL,

`passport\_id` INT NOT NULL,

`adress\_id` MEDIUMINT NOT NULL,

`role\_id` TINYINT NOT NULL,

`status\_id` TINYINT NOT NULL,

`register\_date` DATETIME NOT NULL,

`time` TIMESTAMP NOT NULL,

PRIMARY KEY (`id`))

CREATE TABLE IF NOT EXISTS `app\_schema`.`block\_log` (

`id` MEDIUMINT NOT NULL AUTO\_INCREMENT,

`client\_id` MEDIUMINT NOT NULL,

`start\_time` DATETIME NOT NULL,

`end\_time` DATETIME NOT NULL,

PRIMARY KEY (`id`),

INDEX `fk\_block\_log\_client1\_idx` (`client\_id` ASC),

CONSTRAINT `fk\_block\_log\_client1`

FOREIGN KEY (`client\_id`)

REFERENCES `app\_schema`.`client` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

Триггеры:

DROP TRIGGER insert\_trigger;

CREATE TRIGGER `insert\_trigger` AFTER INSERT ON `client`

FOR EACH ROW

INSERT INTO `insert\_log` SET

user\_id = NEW.id,

email = NEW.email,

login = NEW.login,

password = NEW.password,

phoneNumber = NEW.phoneNumber,

passport\_id = NEW.passport\_id,

adress\_id = NEW.adress\_id,

role\_id = NEW.role\_id,

status\_id = NEW.status\_id,

register\_date = NEW.register\_date;

DROP TRIGGER update\_trigger;

CREATE TRIGGER `update\_trigger` BEFORE UPDATE ON `client`

FOR EACH ROW

INSERT INTO `update\_log` SET

user\_id = OLD.id,

email = OLD.email,

login = OLD.login,

password = OLD.password,

phoneNumber = OLD.phoneNumber,

passport\_id = OLD.passport\_id,

adress\_id = OLD.adress\_id,

role\_id = OLD.role\_id,

status\_id = OLD.status\_id,

register\_date = OLD.register\_date;

DROP TRIGGER delete\_trigger;

CREATE TRIGGER `delete\_trigger` BEFORE DELETE ON `client`

FOR EACH ROW

INSERT INTO `delete\_log` SET

user\_id = OLD.id,

email = OLD.email,

login = OLD.login,

password = OLD.password,

phoneNumber = OLD.phoneNumber,

passport\_id = OLD.passport\_id,

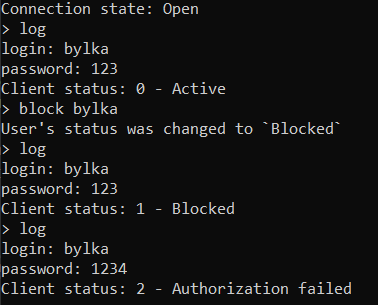
adress\_id = OLD.adress\_id,

role\_id = OLD.role\_id,

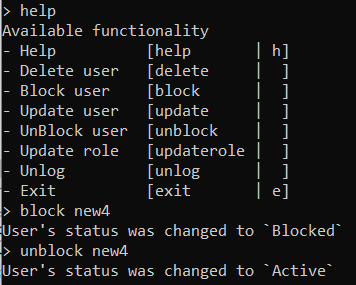
status\_id = OLD.status\_id,

register\_date = OLD.register\_date;

Пример работы программы



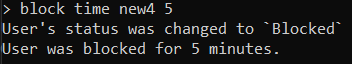
Скриншот 1. Авторизация пользователя

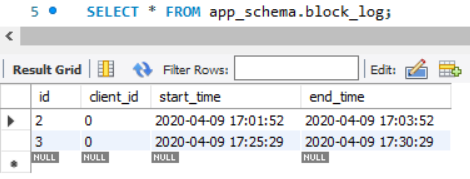


Cкриншот 2. Блокировка / разблокировка пользователя

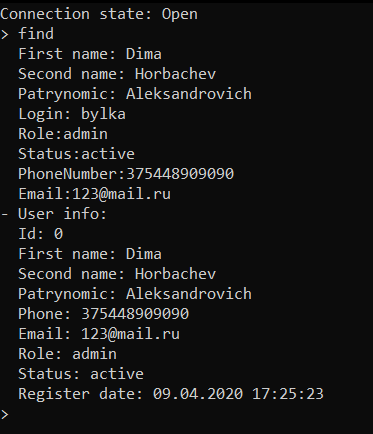


Скриншот 3. Обновление роли пользователя

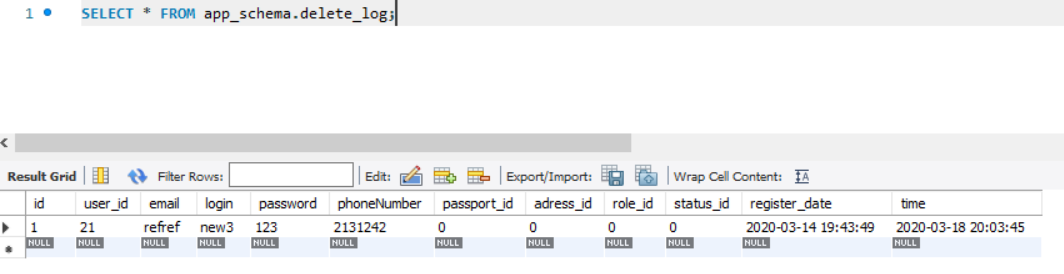




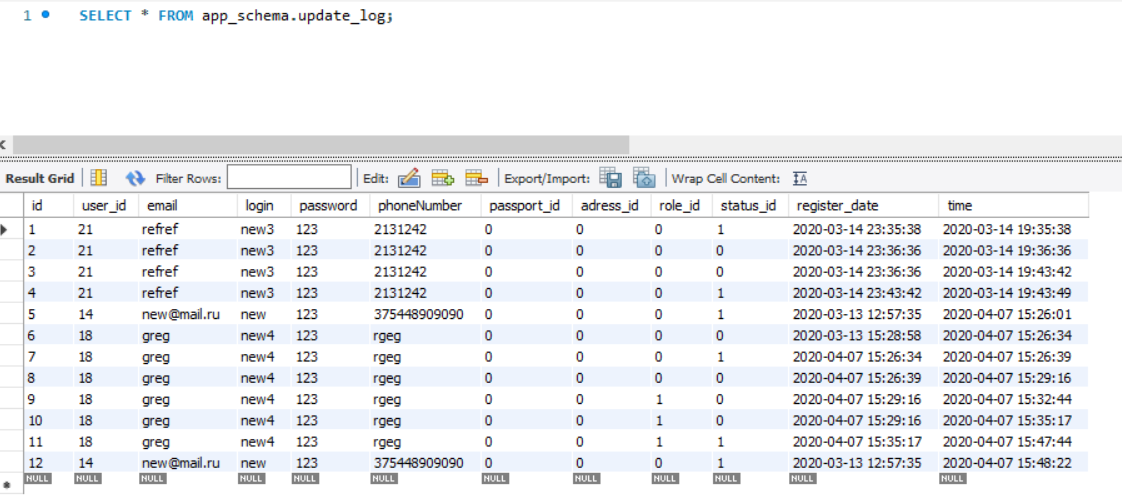
Cкриншот 4. Блокировка пользователя с автоматическим снятием по истечению времени



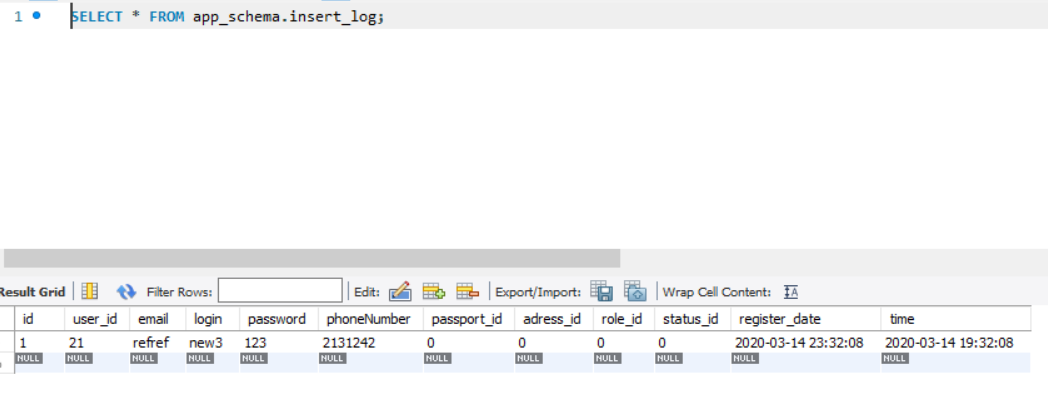
Скриншот 5. Поиск пользователя одним запросом по всем критериям



Скриншот 6. Логи удалений



Скриншот 7. Логи обновлений



Скриншот 8. Логи вставок

Код программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.SqlClient;

using MySqlConnector;

using MySql.Data.MySqlClient;

using System.Data;

using System.Threading;

namespace CarSharingORMApp

{

enum status

{

Active,

Blocked

}

enum role

{

User,

Operator,

Admin

}

class Program

{

private static MySqlConnection currentConnection;

private const string ConnectionString = "server = 192.168.190.130; port=3306;username=student;password=123;database=app\_schema";

private static User currentUser = new Admin(0, 0, 0) { RoleId = 2, Id = 0, StatusId = 0 }; //new UnregistredUser();

private static bool isRunning = true;

static void Main(string[] args)

{

Setup();

OpenConnection();

CheckConnection();

while (isRunning)

{

Console.Write("> ");

HandleCommand(Console.ReadLine());

}

}

private static void Setup()

{

currentConnection = new MySqlConnection(ConnectionString);

}

private static void CheckConnection()

{

Console.WriteLine($"Connection state: {currentConnection.State}");

}

private static void OpenConnection()

{

try

{

currentConnection.Open();

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

}

private static void CloseConnection()

{

currentConnection.Close();

}

private static void UserAuthorization()

{

Console.Write("login: ");

string userName = Console.ReadLine();

Console.Write("password: ");

string password = Console.ReadLine();

DataTable table = new DataTable();

MySqlDataAdapter adapter = new MySqlDataAdapter();

MySqlCommand command = new MySqlCommand("SELECT \* FROM `client` WHERE `login` = @uL AND `password` = @uP", currentConnection);

command.Parameters.Add("@uL", MySqlDbType.VarChar).Value = userName;

command.Parameters.Add("@uP", MySqlDbType.Int64).Value = long.Parse(password);

adapter.SelectCommand = command;

adapter.Fill(table);

if (table.Rows.Count != 0)

{

UnLogging();

currentUser = ((UnregistredUser)currentUser).Authorize(table.Rows[0]["role\_id"].ToString(), table.Rows[0]["status\_id"].ToString(), table.Rows[0]["id"].ToString());

status currentStatus = (status)int.Parse(table.Rows[0]["status\_id"].ToString());

if (currentUser is null)

Console.WriteLine($"Client status: {(int)currentStatus} - {currentStatus}");

Console.WriteLine($"Client status: {(int)currentStatus} - {currentStatus}");

}

else

{

Console.WriteLine("Client status: 2 - Authorization failed");

}

}

private static void UserRegistration()

{

Console.Write("email: ");

string email = Console.ReadLine();

Console.Write("login: ");

string login = Console.ReadLine();

Console.Write("password: ");

string password = Console.ReadLine();

Console.Write("phone number: ");

string phone = Console.ReadLine();

MySqlTransaction sqlTransaction = currentConnection.BeginTransaction();

try

{

MySqlCommand command = new MySqlCommand("LOCK TABLES client WRITE", currentConnection);

command.Transaction = sqlTransaction;

command.ExecuteNonQuery();

command.CommandText = "INSERT INTO client " +

"(email, login, password, phonenumber, passport\_id, adress\_id, role\_id, status\_id) " +

"VALUES " +

"(@uEmail, @uLogin, @uPassword, @uPhone, @uPassport, @uAdress, @uRole, @uStatus)";

command.Parameters.Add("@uEmail", MySqlDbType.VarChar).Value = email;

command.Parameters.Add("@uLogin", MySqlDbType.VarChar).Value = login;

command.Parameters.Add("@uPassword", MySqlDbType.Int32).Value = int.Parse(password);

command.Parameters.Add("@uPhone", MySqlDbType.VarChar).Value = phone;

command.Parameters.Add("@uPassport", MySqlDbType.Int16).Value = 0;

command.Parameters.Add("@uAdress", MySqlDbType.Int16).Value = 0;

command.Parameters.Add("@uStatus", MySqlDbType.VarChar).Value = 0;

command.Parameters.Add("@uRole", MySqlDbType.Int16).Value = 0;

command.ExecuteNonQuery();

command.CommandText = "UNLOCK TABLES";

command.ExecuteNonQuery();

sqlTransaction.Commit();

Console.WriteLine("User registred.");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

sqlTransaction.Rollback();

return;

}

}

private static void DeleteUser(string login)

{

MySqlTransaction sqlTransaction = currentConnection.BeginTransaction();

try

{

MySqlCommand command = currentConnection.CreateCommand();

command.Transaction = sqlTransaction;

command.CommandText = "LOCK TABLES client WRITE";

command.ExecuteNonQuery();

command.CommandText = "SELECT \* FROM `client` WHERE `login` = @uL";

command.Parameters.Add("@uL", MySqlDbType.VarChar).Value = login;

int isExists = command.ExecuteNonQuery();

if (isExists != 0)

{

command.CommandText = "DELETE FROM client WHERE `login` = @uL";

int isDeleted = command.ExecuteNonQuery();

command.CommandText = "UNLOCK TABLES";

command.ExecuteNonQuery();

sqlTransaction.Commit();

if (isDeleted != 0)

Console.WriteLine($"User {login} was deleted.");

}

else

{

sqlTransaction.Rollback();

Console.WriteLine($"User with login `{login}` doesn't exist");

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

sqlTransaction.Rollback();

return;

}

}

private static void BlockUser(string login)

{

try

{

MySqlCommand command = currentConnection.CreateCommand();

command.Parameters.Add("@uL", MySqlDbType.VarChar).Value = login;

command.Parameters.Add("@uS", MySqlDbType.Int16).Value = (int)status.Blocked;

command.CommandText = "UPDATE client SET `status\_id` = @uS WHERE `login` = @uL";

int isChanged = command.ExecuteNonQuery();

if (isChanged != 0)

Console.WriteLine($"User's status was changed to `{status.Blocked}`");

else

{

Console.WriteLine($"User with login `{login}` doesn't exist");

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

return;

}

}

private static void FindUserBy(string parameters)

{

Console.Write(" First name: ");

string firstName = Console.ReadLine();

Console.Write(" Second name: ");

string secondName = Console.ReadLine();

Console.Write(" Patrynomic: ");

string patrynomic = Console.ReadLine();

Console.Write(" Login: ");

string Login = Console.ReadLine();

Console.Write(" Role:");

string role = Console.ReadLine();

Console.Write(" Status:");

string status = Console.ReadLine();

Console.Write(" PhoneNumber:");

string phone = Console.ReadLine();

Console.Write(" Email:");

string email = Console.ReadLine();

StringBuilder stringBuilder = new StringBuilder();

stringBuilder.Append(

$"`login` = '{Login}' " +

$"AND " +

$"`role`.name = '{role}' " +

$"AND " +

$"`status`.name = '{status}' " +

$"AND " +

$"`firstname` = '{firstName}' " +

$"AND " +

$"`secondname` = '{secondName}' " +

$"AND " +

$"patrynomic = '{patrynomic}'" +

$"AND " +

$"`email` = '{email}' " +

$"AND " +

$"phoneNumber = '{phone}'");

try

{

var command = currentConnection.CreateCommand();

command.CommandText = $"SELECT `client`.id, phoneNumber, email, register\_date, patrynomic, firstname, secondname , status.name as status, role.name as role FROM `client` " +

"INNER JOIN `passport` ON `client`.passport\_id = `passport`.id " +

"inner join patrynomic on `passport`.patrynomic\_id = `patrynomic`.id " +

"inner join firstname on `passport`.patrynomic\_id = `firstname`.id " +

"inner join secondname on `passport`.patrynomic\_id = `secondname`.id " +

"inner join status on `client`.status\_id = `status`.id " +

"inner join role on `client`.role\_id = `role`.id " +

$"WHERE {stringBuilder}";

MySqlDataAdapter adapter = new MySqlDataAdapter();

adapter.SelectCommand = command;

DataTable table = new DataTable();

adapter.Fill(table);

DataRow row = table.Rows[0];

if (table.Rows.Count != 0)

{

Console.WriteLine("- User info: ");

Console.WriteLine(" Id: {0}", row["id"]);

Console.WriteLine(" First name: {0}", row["firstname"]);

Console.WriteLine(" Second name: {0}", row["secondname"]);

Console.WriteLine(" Patrynomic: {0}", row["patrynomic"]);

Console.WriteLine(" Phone: {0}", row["phoneNumber"]);

Console.WriteLine(" Email: {0}", row["email"]);

Console.WriteLine(" Role: {0}", row["role"]);

Console.WriteLine(" Status: {0}", row["status"]);

Console.WriteLine(" Register date: {0}", row["register\_date"]);

}

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

}

private static void TimeBlockUser(string login, int time)

{

try

{

MySqlCommand command = new MySqlCommand("LOCK TABLES client WRITE", currentConnection);

command.Parameters.Add("@uL", MySqlDbType.VarChar).Value = login;

command.Parameters.Add("@uS", MySqlDbType.Int16).Value = (int)status.Blocked;

command.ExecuteNonQuery();

command.CommandText = "UPDATE client SET `status\_id` = @uS WHERE `login` = @uL";

int isChanged = command.ExecuteNonQuery();

if (isChanged != 0)

Console.WriteLine($"User's status was changed to `{status.Blocked}`");

else

{

Console.WriteLine($"User with login `{login}` doesn't exist");

}

command.CommandText = "UNLOCK TABLES";

command.ExecuteNonQuery();

DataTable table = new DataTable();

MySqlDataAdapter adapter = new MySqlDataAdapter();

command.CommandText = "SELECT \* FROM `client` WHERE `login` = @uL";

adapter.SelectCommand = command;

adapter.Fill(table);

command.CommandText = $"CREATE EVENT IF NOT EXISTS {login + "\_event"} " +

"ON SCHEDULE AT CURRENT\_TIMESTAMP + INTERVAL @time MINUTE " +

"DO UPDATE client SET `status\_id` = 0 WHERE `login` = @uL";

command.Parameters.Add("@time", MySqlDbType.Int32).Value = time;

command.ExecuteNonQuery();

command.CommandText = $"INSERT INTO block\_log (client\_id, start\_time, end\_time) VALUES ({table.Rows[0]["id"]}, CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP + INTERVAL @time MINUTE)";

command.ExecuteNonQuery();

Console.WriteLine($"User was blocked for {time} minutes.");

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

}

private static void Restore(string login)

{

try

{

var command = currentConnection.CreateCommand();

command.Parameters.Add("@login", MySqlDbType.VarChar).Value = login;

command.CommandText = "SELECT \* FROM `backup` WHERE `login` = @login AND `backup\_creation\_time` = (SELECT MAX(`backup\_creation\_time`) FROM `backup`)";

MySqlDataReader reader = command.ExecuteReader();

if (reader.HasRows)

{

reader.Read();

command.Parameters.Add("@bId", MySqlDbType.Int32).Value = reader["client\_id"];

command.Parameters.Add("@bEmail", MySqlDbType.VarChar).Value = reader["email"];

command.Parameters.Add("@bLogin", MySqlDbType.Int64).Value = reader["login"];

command.Parameters.Add("@bPassword", MySqlDbType.VarChar).Value = reader["password"];

command.Parameters.Add("@bPhone", MySqlDbType.VarChar).Value = reader["phoneNumber"];

command.Parameters.Add("@bPassport", MySqlDbType.Int32).Value = reader["passport\_id"];

command.Parameters.Add("@bAdress", MySqlDbType.Int32).Value = reader["adress\_id"];

command.Parameters.Add("@bRole", MySqlDbType.Int32).Value = reader["role\_id"];

command.Parameters.Add("@bStatus", MySqlDbType.Int32).Value = reader["status\_id"];

command.Parameters.Add("@bRegDate", MySqlDbType.DateTime).Value = reader["register\_date"];

object backupTime = reader["backup\_creation\_time"];

string operation = reader["operation"].ToString();

Console.WriteLine($"- Last backup info for {login}:\n Operation: {operation}\n Time: {backupTime}");

reader.Close();

command.CommandText = "LOCK TABLES client WRITE";

command.ExecuteNonQuery();

if (operation == "update")

command.CommandText =

"UPDATE " +

"client " +

"SET " +

"`id` = @bId, `email` = @bEmail, `login` = @bLogin, `password` = @bPassword, " +

"`phoneNumber` = @bPhone, `passport\_id` = @bPassport, `adress\_id` = @bAdress, `role\_id` = @bRole, " +

"`status\_id` = @bStatus, `register\_date` = @bRegDate WHERE `login` = @login";

if (operation == "delete")

command.CommandText =

"INSERT " +

"client " +

"SET " +

"`id` = @bId, `email` = @bEmail, `login` = @bLogin, `password` = @bPassword, " +

"`phoneNumber` = @bPhone, `passport\_id` = @bPassport, `adress\_id` = @bAdress, `role\_id` = @bRole, " +

"`status\_id` = @bStatus, `register\_date` = @bRegDate";

int isUpdated = command.ExecuteNonQuery();

if (isUpdated > 0)

{

Console.WriteLine($"- User {login} was restored. ");

}

command.CommandText = "UNLOCK TABLES";

command.ExecuteNonQuery();

}

else

{

reader.Close();

Console.WriteLine($"- No backup for {login}");

}

}

catch(Exception ex)

{

Console.WriteLine("Restoring failed: {0}", ex.Message);

}

}

private static void UpdateRole(string login, int role)

{

try

{

var command = currentConnection.CreateCommand();

command.Parameters.Add("@role",MySqlDbType.Int32).Value = role;

command.Parameters.Add("@login", MySqlDbType.VarChar).Value = login;

command.CommandText = "LOCK TABLES client WRITE";

command.ExecuteNonQuery();

command.CommandText = "UPDATE client SET `role\_id` = @role WHERE `login` = @login";

int isUpdated = command.ExecuteNonQuery();

if(isUpdated != 0)

{

Console.WriteLine($"User's role was updated to {(role)role}");

}

command.CommandText = "UNLOCK TABLES";

command.ExecuteNonQuery();

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

}

private static void UnBlockUser(string login)

{

try

{

MySqlCommand command = new MySqlCommand("LOCK TABLES client WRITE", currentConnection);

command.ExecuteNonQuery();

command.CommandText ="UPDATE client SET `status\_id` = @uS WHERE `login` = @uL";

command.Parameters.Add("@uL", MySqlDbType.VarChar).Value = login;

command.Parameters.Add("@uS", MySqlDbType.Int16).Value = (int)status.Active;

command.ExecuteNonQuery();

Console.WriteLine($"User's status was changed to `{status.Active}`");

command.CommandText = "UNLOCK TABLES";

command.ExecuteNonQuery();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

return;

}

}

private static void AvailableCommands()

{

Console.WriteLine("Available functionality");

Console.WriteLine("- Help [help | h]");

if (currentUser is UnregistredUser)

{

Console.WriteLine("- Registration [registrare | r]");

Console.WriteLine("- Authorization [log | a]");

}

if (currentUser is Admin && (((Admin)currentUser).StatusId != (int)status.Blocked))

{

Console.WriteLine("- Delete user [delete | ]");

Console.WriteLine("- Block user [block | ]");

Console.WriteLine("- Update user [update | ]");

Console.WriteLine("- UnBlock user [unblock | ]");

Console.WriteLine("- Update role [updaterole | ]");

}

Console.WriteLine("- Unlog [unlog | ]");

Console.WriteLine("- Exit [exit | e] ");

}

private static void UnLogging()

{

currentUser = new UnregistredUser();

}

private static void HandleCommand(string command)

{

try

{

string[] splittedCommand = command.Split(' ');

if (string.Compare(command, "log", true) == 0 || string.Compare(command, "a", true) == 0 && currentUser is UnregistredUser)

{

UserAuthorization();

}

if (string.Compare(command, "help", true) == 0 || string.Compare(command, "h", true) == 0)

{

AvailableCommands();

}

if (string.Compare(command, "register", true) == 0 || string.Compare(command, "r", true) == 0 && currentUser is UnregistredUser)

{

UserRegistration();

}

if (string.Compare(command, "exit", true) == 0 || string.Compare(command, "e", true) == 0)

{

CloseConnection();

CheckConnection();

isRunning = false;

}

if (string.Compare(command, "unlog", true) == 0 || string.Compare(command, "r", true) == 0 && currentUser is UnregistredUser)

{

UnLogging();

}

if (string.Compare(splittedCommand[0], "block", true) == 0 && currentUser is Admin)

{

if (string.Compare(splittedCommand[1], "time", true) == 0)

TimeBlockUser(splittedCommand[2], int.Parse(splittedCommand[3]));

else

BlockUser(splittedCommand[1]);

}

if (string.Compare(splittedCommand[0], "unblock", true) == 0 && currentUser is Admin)

{

UnBlockUser(splittedCommand[1]);

}

if (string.Compare(splittedCommand[0], "delete", true) == 0 && currentUser is Admin)

{

DeleteUser(splittedCommand[1]);

}

if (string.Compare(splittedCommand[0], "updaterole", true) == 0 && currentUser is Admin)

{

switch(splittedCommand[2])

{

case "operator":

UpdateRole(splittedCommand[1], (int)role.Operator);

break;

case "admin":

UpdateRole(splittedCommand[1], (int)role.Admin);

break;

case "user":

UpdateRole(splittedCommand[1], (int)role.User);

break;

default: break;

}

}

if (string.Compare(splittedCommand[0], "find", true) == 0 && currentUser is Admin)

{

if (splittedCommand.Length < 2)

FindUserBy(null);

else

FindUserBy(splittedCommand[1]);

}

if (string.Compare(splittedCommand[0], "restore", true) == 0)

{

Restore(splittedCommand[1]);

}

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}