

Министерство образования Республики Беларусь  
Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №2  
по дисциплине «Модели данных и системы управления базами данных»  
по теме «Интернет площадка услуг по аренде легковых  
автомобилями(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

```
Connection state: Open  
> log  
login: bylka  
password: 123  
Client status: 0 - Active  
> block bylka  
User's status was changed to `Blocked`  
> log  
login: bylka  
password: 123  
Client status: 1 - Blocked  
> log  
login: bylka  
password: 1234  
Client status: 2 - Authorization failed
```

Скриншот 1. Авторизация пользователя

```
> help
Available functionality
- Help          [help      | h]
- Delete user   [delete   |  ]
- Block user    [block    |  ]
- Update user   [update   |  ]
- Unblock user  [unblock  |  ]
- Update role   [updaterole |  ]
- Unlog         [unlog     |  ]
- Exit         [exit      | e]
> block new4
User's status was changed to `Blocked`
> unblock new4
User's status was changed to `Active`
```

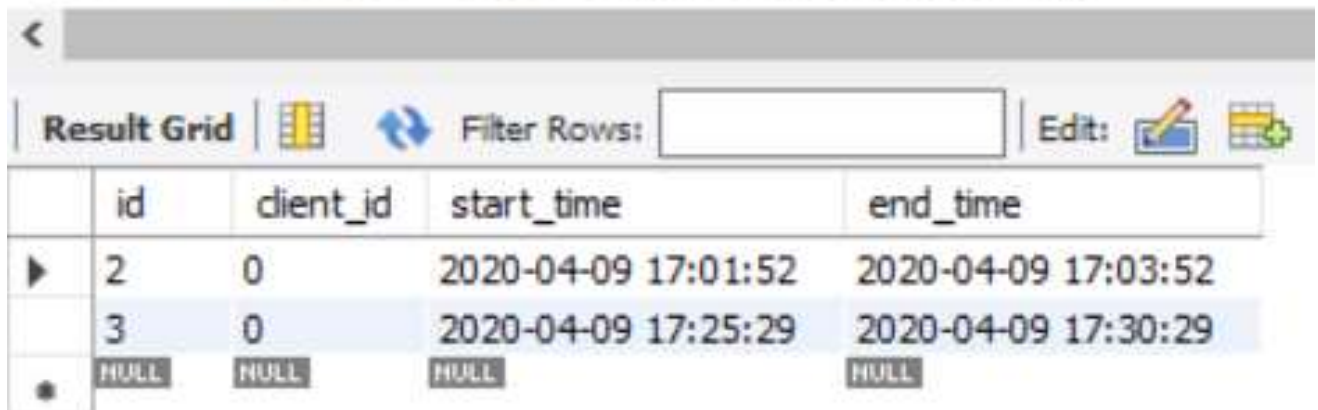
Скриншот 2. Блокировка / разблокировка пользователя

```
> updaterole new4 operator
User's role was updated to Operator
```

Скриншот 3. Обновление роли пользователя

```
> block time new4 5
User's status was changed to `Blocked`
User was blocked for 5 minutes.
```

```
5 • SELECT * FROM app_schema.block_log;
```



The screenshot shows a database query result grid. The grid has a toolbar with icons for 'Result Grid', 'Filter Rows', and 'Edit'. The query executed is 'SELECT \* FROM app\_schema.block\_log;'. The result is a table with 5 columns: 'id', 'client\_id', 'start\_time', and 'end\_time'. There are 3 rows of data. The first row has id 2, client\_id 0, start\_time 2020-04-09 17:01:52, and end\_time 2020-04-09 17:03:52. The second row has id 3, client\_id 0, start\_time 2020-04-09 17:25:29, and end\_time 2020-04-09 17:30:29. The third row has NULL values for all columns.

	id	client_id	start_time	end_time
▶	2	0	2020-04-09 17:01:52	2020-04-09 17:03:52
	3	0	2020-04-09 17:25:29	2020-04-09 17:30:29
●	NULL	NULL	NULL	NULL

Скриншот 4. Блокировка пользователя с автоматическим снятием по истечению времени

```
Connection state: Open
> find
  First name: Dima
  Second name: Horbachev
  Patronymic: Aleksandrovich
  Login: bylka
  Role: admin
  Status: active
  PhoneNumber: 375448909090
  Email: 123@mail.ru
- User info:
  Id: 0
  First name: Dima
  Second name: Horbachev
  Patronymic: Aleksandrovich
  Phone: 375448909090
  Email: 123@mail.ru
  Role: admin
  Status: active
  Register date: 09.04.2020 17:25:23
>
```

Скриншот 5. Поиск пользователя одним запросом по всем критериям



```
1 • SELECT * FROM app_schema.delete_log;
```

Result Grid												
Filter Rows:												
Edit:												
Export/Import:												
Wrap Cell Content:												
	id	user_id	email	login	password	phoneNumber	passport_id	adress_id	role_id	status_id	register_date	time
▶	1	21	refref	new3	123	2131242	0	0	0	0	2020-03-14 19:43:49	2020-03-18 20:03:45
*												







Скриншот 6. Логи удалений

```
1 • SELECT * FROM app_schema.update_log;
```

Result Grid												
Filter Rows:												
Edit:												
Export/Import:												
Wrap Cell Content:												
	id	user_id	email	login	password	phoneNumber	passport_id	adress_id	role_id	status_id	register_date	time
▶	1	21	refref	new3	123	2131242	0	0	0	1	2020-03-14 23:35:38	2020-03-14 19:35:38
	2	21	refref	new3	123	2131242	0	0	0	0	2020-03-14 23:36:36	2020-03-14 19:36:36
	3	21	refref	new3	123	2131242	0	0	0	0	2020-03-14 23:36:36	2020-03-14 19:43:42
	4	21	refref	new3	123	2131242	0	0	0	1	2020-03-14 23:43:42	2020-03-14 19:43:49
	5	14	new@mail.ru	new	123	375448909090	0	0	0	1	2020-03-13 12:57:35	2020-04-07 15:26:01
	6	18	greg	new4	123	rgeg	0	0	0	0	2020-03-13 15:28:58	2020-04-07 15:26:34
	7	18	greg	new4	123	rgeg	0	0	0	1	2020-04-07 15:26:34	2020-04-07 15:26:39
	8	18	greg	new4	123	rgeg	0	0	0	0	2020-04-07 15:26:39	2020-04-07 15:29:16
	9	18	greg	new4	123	rgeg	0	0	1	0	2020-04-07 15:29:16	2020-04-07 15:32:44
	10	18	greg	new4	123	rgeg	0	0	1	0	2020-04-07 15:29:16	2020-04-07 15:35:17
	11	18	greg	new4	123	rgeg	0	0	1	1	2020-04-07 15:35:17	2020-04-07 15:47:44
	12	14	new@mail.ru	new	123	375448909090	0	0	0	1	2020-03-13 12:57:35	2020-04-07 15:48:22
*												

Скриншот 7. Логи обновлений

```
1 • SELECT * FROM app_schema.insert_log;
```

Result Grid											
Filter Rows: <input type="text"/>											
Edit:      Export/Import:     Wrap Cell Content: 											
id	user_id	email	login	password	phoneNumber	passport_id	adress_id	role_id	status_id	register_date	time
1	21	refref	new3	123	2131242	0	0	0	0	2020-03-14 23:32:08	2020-03-14 19:32:08
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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();

    MySqlConnection 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)
{
    MySqlConnection 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(" Patronymic: ");
string patronymic = 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 " +
    $"`patronymic` = '{patronymic}' " +
    $"AND " +
    $"`email` = '{email}' " +
    $"AND " +
    $"`phoneNumber` = '{phone}'");

try
{
    var command = currentConnection.CreateCommand();
    command.CommandText = $"SELECT `client`.id, phoneNumber, email, register_date,
patronymic, firstname, secondname , status.name as status, role.name as role FROM `client` " +
        "INNER JOIN `passport` ON `client`.passport_id = `passport`.id " +
        "inner join patronymic on `passport`.patronymic_id = `patronymic`.id " +
        "inner join firstname on `passport`.patronymic_id = `firstname`.id " +
        "inner join secondname on `passport`.patronymic_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(" Patronymic: {0}", row["patronymic"]);
    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();
        }
    }
    catch { }
}

```

```

    }
    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);
}
}
}

```