



Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИУК «Информатика и управление»**

КАФЕДРА **ИУК4 «Программное обеспечение ЭВМ и информационные технологии»**

ЛАБОРАТОРНАЯ РАБОТА 5

**ДИСЦИПЛИНА: «ПРОЕКТИРОВАНИЕ СИСТЕМ ХРАНЕНИЯ И
ОБРАБОТКИ ДАННЫХ»**

Выполнил: студент гр. ИУК4-52Б _____ (____ Губин Е.В.____)
(Подпись) (Ф.И.О.)

Проверил: _____ (____ Глебов С.А.____)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:

- Оценка:

Калуга, 2024

Создание триггеров и функций:

1) Создание нового тарифного плана

```
CREATE OR REPLACE PROCEDURE create_new_traffic_plan
(new_title VARCHAR(50),
new_monthly_cost REAL,
new_internet_data_limit REAL,
new_call_minutes_limit INT,
new_messages_limit INT)
LANGUAGE plpgsql
AS
$$
BEGIN
    INSERT INTO traffic_plans (
        title,
        monthly_cost,
        internet_data_limit,
        call_minutes_limit,
        messages_limit
    )
    VALUES
    (
        new_title,
        new_monthly_cost,
        new_internet_data_limit,
        new_call_minutes_limit,
        new_messages_limit
    );
END;
$$;
```

2) Функция, проверяющая принадлежит ли номер телефона клиенту компании

```
CREATE OR REPLACE FUNCTION is_our_client(new_phone_number VARCHAR(50))
RETURNS BOOL
LANGUAGE plpgsql
AS
$$
BEGIN
    RETURN EXISTS (SELECT * FROM sim_cards WHERE phone_number =
new_phone_number);
END;
$$;
```

3) Функция, возвращающая id записи из таблицы связи номеров телефона клиент-клиент

```
CREATE OR REPLACE FUNCTION get_customer_to_customer_id
(new_initiator_phone_number VARCHAR(50),
new_interlocutor_phone_number VARCHAR(50))
RETURNS INT
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN (SELECT customer_to_customer_id FROM customer_to_customer
WHERE initiator_phone_number = new_initiator_phone_number
AND interlocutor_phone_number = new_interlocutor_phone_number);
END;
$$;
```

4) Функция, возвращающая id записи из таблицы связи номеров телефона клиент-клиент другой компании

```
CREATE OR REPLACE FUNCTION get_customer_to_other_id
(new_initiator_phone_number VARCHAR(50),
new_interlocutor_phone_number VARCHAR(50),
new_customer_is_initiator BOOL)
RETURNS INT
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN (SELECT customer_to_other_id
            FROM customer_to_other
            WHERE customer_phone_number = new_interlocutor_phone_number AND
                  other_phone_number = new_initiator_phone_number AND customer_is_initiator
            = new_customer_is_initiator);
END;
$$;
```

5) Процедура, вставляющая новое сообщение в зависимости от типа связи

```
CREATE OR REPLACE PROCEDURE insert_message
(new_message_content VARCHAR(255),
new_message_date_time TIMESTAMP,
new_customer_to_customer_id INT,
new_customer_to_orher_id INT)
LANGUAGE plpgsql
AS
$$
BEGIN
    IF (new_customer_to_orher_id IS NULL) THEN
        INSERT INTO messages (message_content, message_date_time,
customer_to_customer_id)
        VALUES (new_message_content, new_message_date_time,
new_customer_to_customer_id);
    ELSE
        INSERT INTO messages (message_content, message_date_time,
customer_to_other_id)
        VALUES (new_message_content, new_message_date_time,
new_customer_to_orher_id);
    END IF;
END;
$$;
```

6) Процедура создания нового сообщения

```
CREATE OR REPLACE PROCEDURE new_message
(new_message_content VARCHAR(255),
new_message_date_time TIMESTAMP,
new_initiator_phone_number VARCHAR(50),
new_interlocutor_phone_number VARCHAR(50))
LANGUAGE plpgsql
AS
$$
DECLARE
    new_id INT;
BEGIN
    IF (SELECT is_our_client(new_initiator_phone_number))
    AND (SELECT is_our_client(new_interlocutor_phone_number))
    THEN
        IF EXISTS
        (SELECT * FROM customer_to_customer
        WHERE initiator_phone_number = new_initiator_phone_number
        AND interlocutor_phone_number = new_interlocutor_phone_number)
    THEN
```

```

        SELECT
get_customer_to_customer_id(new_initiator_phone_number,
new_interlocutor_phone_number) INTO new_id;

        CALL insert_message(new_message_content,
new_message_date_time, new_id, NULL);
    ELSE
        INSERT INTO customer_to_customer (initiator_phone_number,
interlocutor_phone_number)
VALUES (new_initiator_phone_number,
new_interlocutor_phone_number);

        SELECT
get_customer_to_customer_id(new_initiator_phone_number,
new_interlocutor_phone_number) INTO new_id;

        CALL insert_message(new_message_content,
new_message_date_time, new_id, NULL);
    END IF;
    ELSIF (SELECT is_our_client(new_initiator_phone_number))
OR (SELECT is_our_client(new_interlocutor_phone_number)) THEN

        IF (SELECT is_our_client(new_initiator_phone_number)) THEN
            IF EXISTS (SELECT * FROM customer_to_other
WHERE customer_phone_number = new_initiator_phone_number AND
other_phone_number = new_interlocutor_phone_number AND
customer_is_initiator = TRUE) THEN
                SELECT get_customer_to_other_id
(new_initiator_phone_number,
new_interlocutor_phone_number,
TRUE) INTO new_id;

                CALL insert_message(new_message_content,
new_message_date_time, NULL, new_id);
            ELSE
                INSERT INTO customer_to_other (customer_is_initiator,
other_phone_number, customer_phone_number)
VALUES (TRUE, new_interlocutor_phone_number,
new_initiator_phone_number);

                SELECT get_customer_to_other_id
(new_initiator_phone_number,
new_interlocutor_phone_number,
TRUE) INTO new_id;

                CALL insert_message(new_message_content,
new_message_date_time, NULL, new_id);
            END IF;
        ELSE
            IF EXISTS (SELECT * FROM customer_to_other
WHERE customer_phone_number = new_interlocutor_phone_number
AND
other_phone_number = new_initiator_phone_number AND
customer_is_initiator = FALSE) THEN
                SELECT get_customer_to_other_id
(new_initiator_phone_number,
new_interlocutor_phone_number,
FALSE) INTO new_id;

                CALL insert_message(new_message_content,
new_message_date_time, NULL, new_id);
            ELSE
                INSERT INTO customer_to_other (customer_is_initiator,
other_phone_number, customer_phone_number)
VALUES (FALSE, new_initiator_phone_number,
new_interlocutor_phone_number);

                SELECT get_customer_to_other_id
(new_initiator_phone_number,
new_interlocutor_phone_number,
FALSE) INTO new_id;

```

```

CALL insert_message(new_message_content,
new_message_date_time, NULL, new_id);
END IF;
END IF;
END IF;
END;
$$;

```

- 7) Триггерная функция и триггер, который добавляет бонусные 500 рублей на баланс, если клиент сделал sim-карту в определённый период

```

CREATE OR REPLACE FUNCTION add_to_balance()
RETURNS TRIGGER
LANGUAGE plpgsql
AS
$$
BEGIN
    IF (NEW.balance - OLD.balance >= 2000) THEN
        UPDATE sim_cards SET balance = NEW.balance + 500
        WHERE phone_number = OLD.phone_number;
    END IF;
    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER add_bonus
AFTER UPDATE ON sim_cards
FOR EACH ROW
EXECUTE FUNCTION add_to_balance();

```

- 8) Триггерная функция и триггер, который генерирует новый номер телефона при создании новой sim-карты

```

CREATE OR REPLACE FUNCTION genetate_phone_number()
RETURNS TRIGGER
LANGUAGE plpgsql
AS
$$
DECLARE
    gen_phone_number VARCHAR(50);
BEGIN
    LOOP
        gen_phone_number := '8' || FLOOR(RANDOM() * 89999 + 10000)::TEXT
        || FLOOR(RANDOM() * 89999 + 10000)::TEXT;
        IF NOT EXISTS (SELECT * FROM sim_cards WHERE phone_number =
gen_phone_number) THEN
            NEW.phone_number := gen_phone_number;
            EXIT;
        END IF;
    END LOOP;
    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER new_sim_card_with_generate_phone_number
BEFORE INSERT ON sim_cards
FOR EACH ROW
EXECUTE FUNCTION genetate_phone_number();

```

Результаты выполнения работы:

```
CALL create_new_traffic_plan('HomeMy', 499.99, 20.0, 700, 50);
SELECT * FROM traffic_plans;
```

	traffic_plan_id [PK] integer	title character varying (50)	monthly_cost real	internet_data_limit real	call_minutes_limit integer	messages_limit integer
1	1	Базовый	300	5000	300	100
2	2	Стандартный	500	10000	600	200
3	3	Премиум	700	15000	1200	300
4	4	Home	499.99	20	700	50
5	5	HomeMy	499.99	20	700	50

Рисунок 1 Добавление тарифного плана

```
CALL new_message('Content', '2024-10-06 14:49:23', '89007654321', '89229060764');
CALL new_message('Contentdadsa', '2024-10-06 14:49:23', '89229060764', '89007654321');
CALL new_message('Content to other', '2024-10-06 14:49:23', '89007654321', '89001234567');
CALL new_message('Content to cust', '2024-10-06 14:49:23', '89001234567', '89007654321');
SELECT * FROM messages;
```

	traffic_plan_id [PK] integer	title character varying (50)	monthly_cost real	internet_data_limit real	call_minutes_limit integer	messages_limit integer
1	1	Базовый	300	5000	300	100
2	2	Стандартный	500	10000	600	200
3	3	Премиум	700	15000	1200	300
4	4	Home	499.99	20	700	50
5	5	HomeMy	499.99	20	700	50

Рисунок 2 Добавление сообщений

```
UPDATE sim_cards SET balance = 6500 WHERE phone_number = '89001234567';
SELECT * from sim_cards;
```

	phone_number [PK] character varying (50)	registration_date date	balance real	traffic_plan_id integer	personal_account_number bigint
1	89007654321	2023-02-01	300	2	1000000002
2	89009876543	2023-03-01	230	3	1000000003
3	87227014735	2023-03-01	230	3	1000000003
4	89001234567	2023-01-01	7000	1	1000000001

Рисунок 3 Бонусные 500 рублей при пополнении от 2000 рублей

```
INSERT INTO sim_cards (registration_date, balance, traffic_plan_id, personal_account_number)
VALUES
('2023-03-01', 234.00, 2, 1000000004);
SELECT * FROM sim_cards;
```

	phone_number [PK] character varying (50)	registration_date date	balance real	traffic_plan_id integer	personal_account_number bigint
1	89007654321	2023-02-01	300	2	1000000002
2	89009876543	2023-03-01	230	3	1000000003
3	87227014735	2023-03-01	230	3	1000000003
4	89001234567	2023-01-01	7000	1	1000000001
5	81692377712	2023-03-01	234	2	1000000003

Рисунок 4 Генерация номера телефона при добавлении новой sim-карты

Выводы: в ходе выполнения лабораторной работы были получены навыки и практический опыт написания функций, процедур и триггеров.



Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИУК «Информатика и управление»**

КАФЕДРА **ИУК4 «Программное обеспечение ЭВМ и информационные технологии»**

ЛАБОРАТОРНАЯ РАБОТА 6

**ДИСЦИПЛИНА: «ПРОЕКТИРОВАНИЕ СИСТЕМ ХРАНЕНИЯ И
ОБРАБОТКИ ДАННЫХ»**

Выполнил: студент гр. ИУК4-52Б _____ (____ Губин Е.В.____)
(Подпись) (Ф.И.О.)

Проверил: _____ (____ Глебов С.А.____)
(Подпись) (Ф.И.О.)

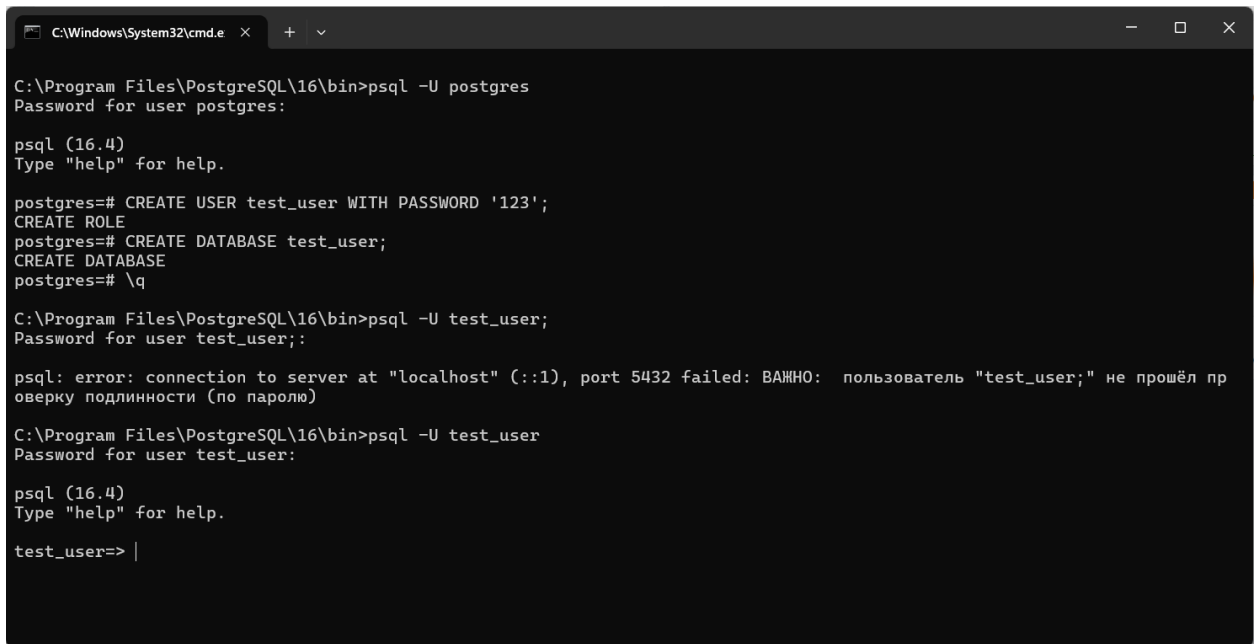
Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:

- Оценка:

Калуга , 2024



```
C:\Windows\System32\cmd.e x + v

C:\Program Files\PostgreSQL\16\bin>psql -U postgres
Password for user postgres:

psql (16.4)
Type "help" for help.

postgres=# CREATE USER test_user WITH PASSWORD '123';
CREATE ROLE
postgres=# CREATE DATABASE test_user;
CREATE DATABASE
postgres=# \q

C:\Program Files\PostgreSQL\16\bin>psql -U test_user;
Password for user test_user;:

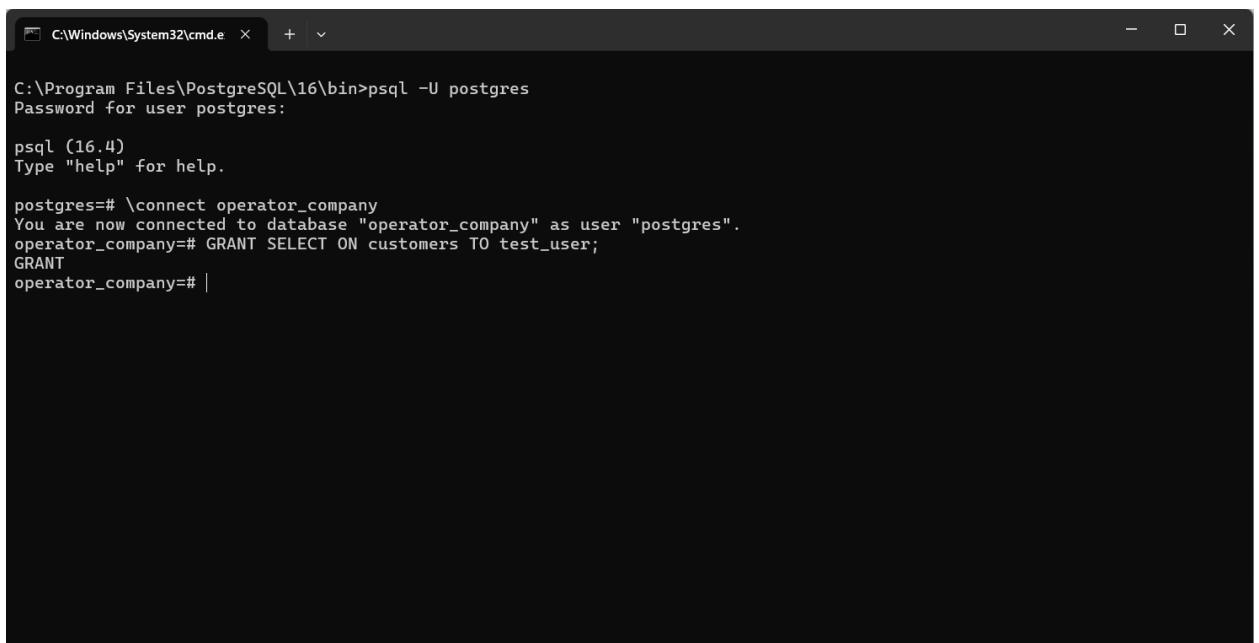
psql: error: connection to server at "localhost" (::1), port 5432 failed: ВАЖНО: пользователь "test_user;" не прошёл пр
оверку подлинности (но паролю)

C:\Program Files\PostgreSQL\16\bin>psql -U test_user
Password for user test_user:

psql (16.4)
Type "help" for help.

test_user=> |
```

Рисунок 1 Создание пользователя



```
C:\Windows\System32\cmd.e x + v

C:\Program Files\PostgreSQL\16\bin>psql -U postgres
Password for user postgres:

psql (16.4)
Type "help" for help.

postgres=# \connect operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# GRANT SELECT ON customers TO test_user;
GRANT
operator_company=# |
```

Рисунок 2 Добавление привилегий на запрос SELECT

```
C:\Windows\System32\cmd.e x + v
C:\Program Files\PostgreSQL\16\bin>psql -U test_user
Password for user test_user:

psql (16.4)
Type "help" for help.

test_user=> \connect operator_company
You are now connected to database "operator_company" as user "test_user".
operator_company=> SELECT * FROM customers;
personal_account_number | last_name | first_name | patronymic | email | birth_date | hashed_password
-----
1000000002 | Петров | Петр | Петрович | petrov@example.com | 1985-05-20 | hashed_password_2
1000000003 | Сидорова | Анна | Сергеевна | sidorova@example.com | 1992-03-10 | hashed_password_3
1000000001 | Губин | Иван | Иванович | ivanov@example.com | 1990-01-15 | hashed_password_1
(3 rows)

operator_company=> UPDATE customers SET email = "ivanov@mail.com" where first_name = 'Иван';
ОШИБКА: столбец "ivanov@mail.com" не существует
LINE 1: UPDATE customers SET email = "ivanov@mail.com" where first_n...
                                ^
operator_company=> UPDATE customers SET email = 'ivanov@mail.com' where first_name = 'Иван';
ОШИБКА: нет доступа к таблице customers
operator_company=> |
```

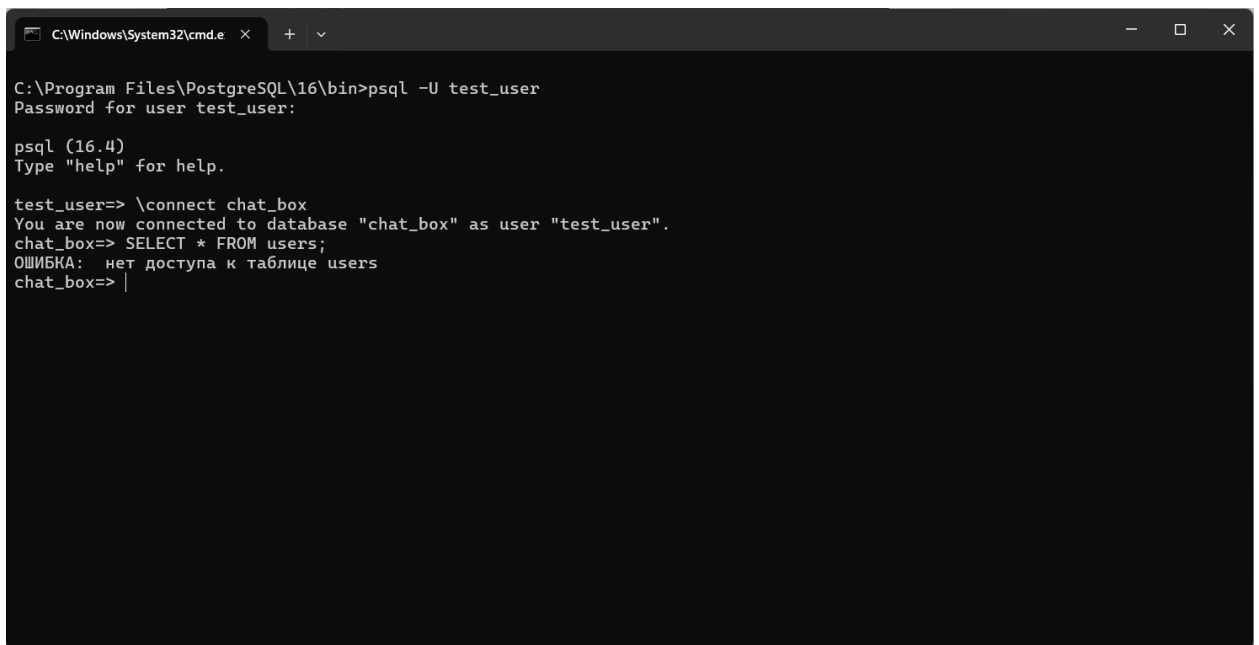
Рисунок 3 Попытка UPDATE test_user

```
C:\Windows\System32\cmd.e x + v
C:\Program Files\PostgreSQL\16\bin>psql -U postgres
Password for user postgres:

psql (16.4)
Type "help" for help.

postgres=# REVOKE CONNECT ON DATABASE chat_box FROM test_user;
REVOKE
postgres=# |
```

Рисунок 4 Запрет на подключение к базе данных chat_box

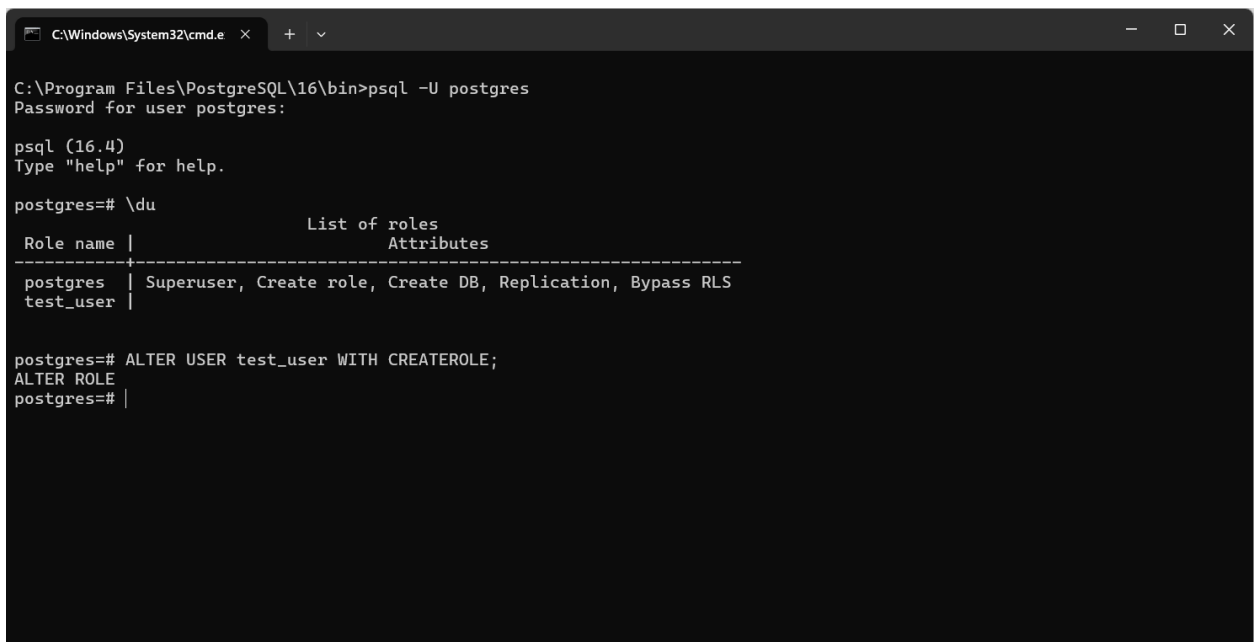


```
C:\Windows\System32\cmd.e x + v
C:\Program Files\PostgreSQL\16\bin>psql -U test_user
Password for user test_user:

psql (16.4)
Type "help" for help.

test_user=> \connect chat_box
You are now connected to database "chat_box" as user "test_user".
chat_box=> SELECT * FROM users;
ОШИБКА: нет доступа к таблице users
chat_box=> |
```

Рисунок 5 Попытка подключения к chat_box



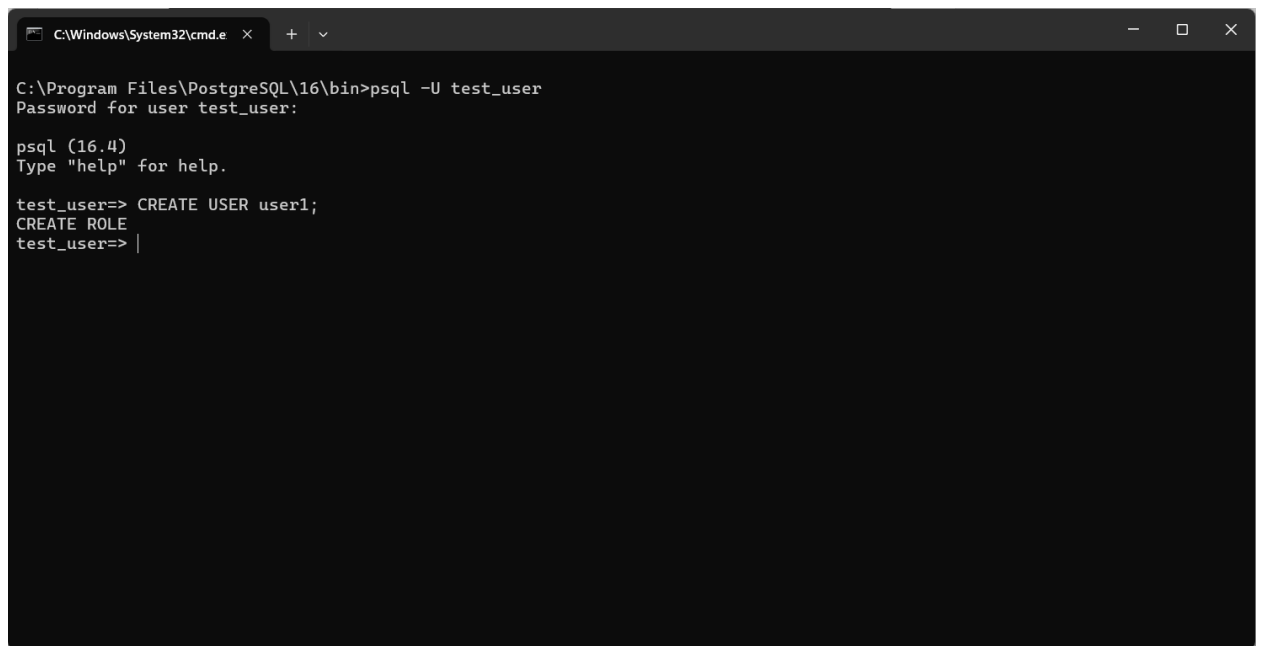
```
C:\Windows\System32\cmd.e x + v
C:\Program Files\PostgreSQL\16\bin>psql -U postgres
Password for user postgres:

psql (16.4)
Type "help" for help.

postgres=# \du
List of roles
Role name | Attributes
-----+-----
postgres | Superuser, Create role, Create DB, Replication, Bypass RLS
test_user |

postgres=# ALTER USER test_user WITH CREATEROLE;
ALTER ROLE
postgres=# |
```

Рисунок 6 Добавление test_user на создание ролей



```
C:\Windows\System32\cmd.e  x  +  v

C:\Program Files\PostgreSQL\16\bin>psql -U test_user
Password for user test_user:

psql (16.4)
Type "help" for help.

test_user=> CREATE USER user1;
CREATE ROLE
test_user=> |
```

Рисунок 7 Создание ролей test_user

Выводы: в ходе выполнения лабораторной работы были получены навыки по разделению прав доступа к данным.



Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИУК «Информатика и управление»**

КАФЕДРА **ИУК4 «Программное обеспечение ЭВМ и информационные технологии»**

ЛАБОРАТОРНАЯ РАБОТА 7

**ДИСЦИПЛИНА: «ПРОЕКТИРОВАНИЕ СИСТЕМ ХРАНЕНИЯ И
ОБРАБОТКИ ДАННЫХ»**

Выполнил: студент гр. ИУК4-52Б _____ (____ Губин Е.В.____)
(Подпись) (Ф.И.О.)

Проверил: _____ (____ Глебов С.А.____)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

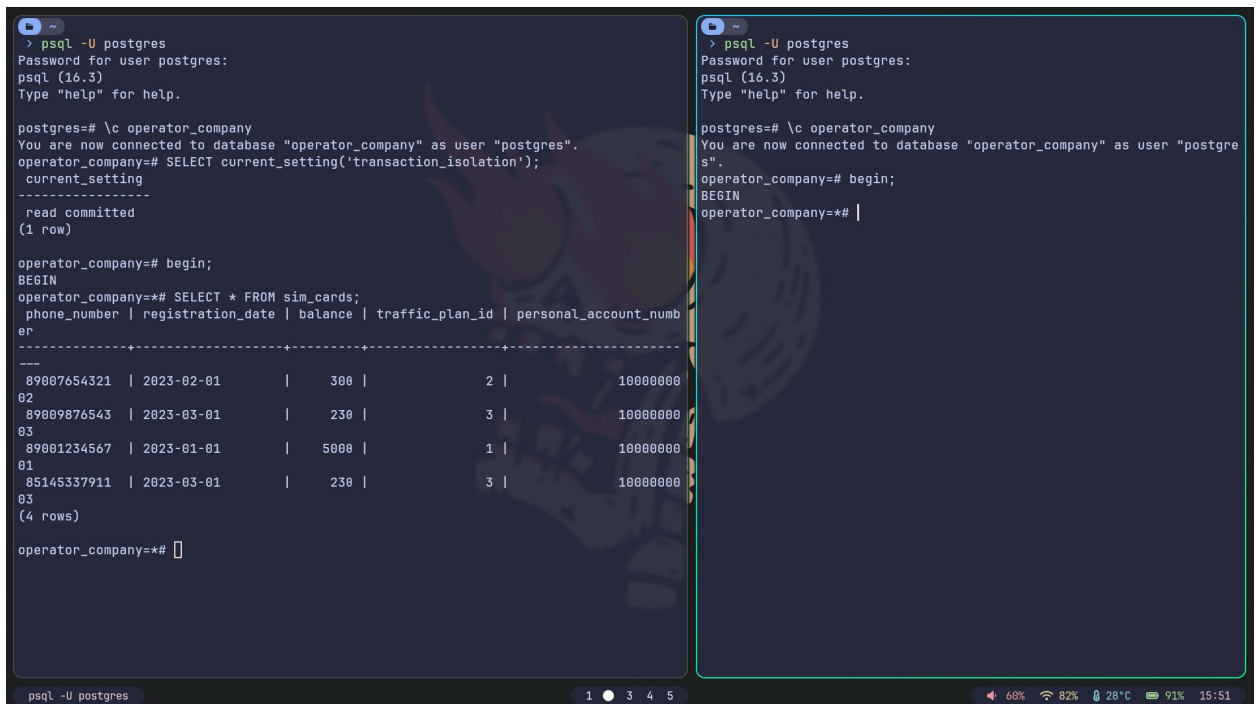
Результаты сдачи (защиты):

- Балльная оценка:

- Оценка:

Калуга , 2024

READ COMMITTED



```
> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# SELECT current_setting('transaction_isolation');
current_setting
-----
read committed
(1 row)

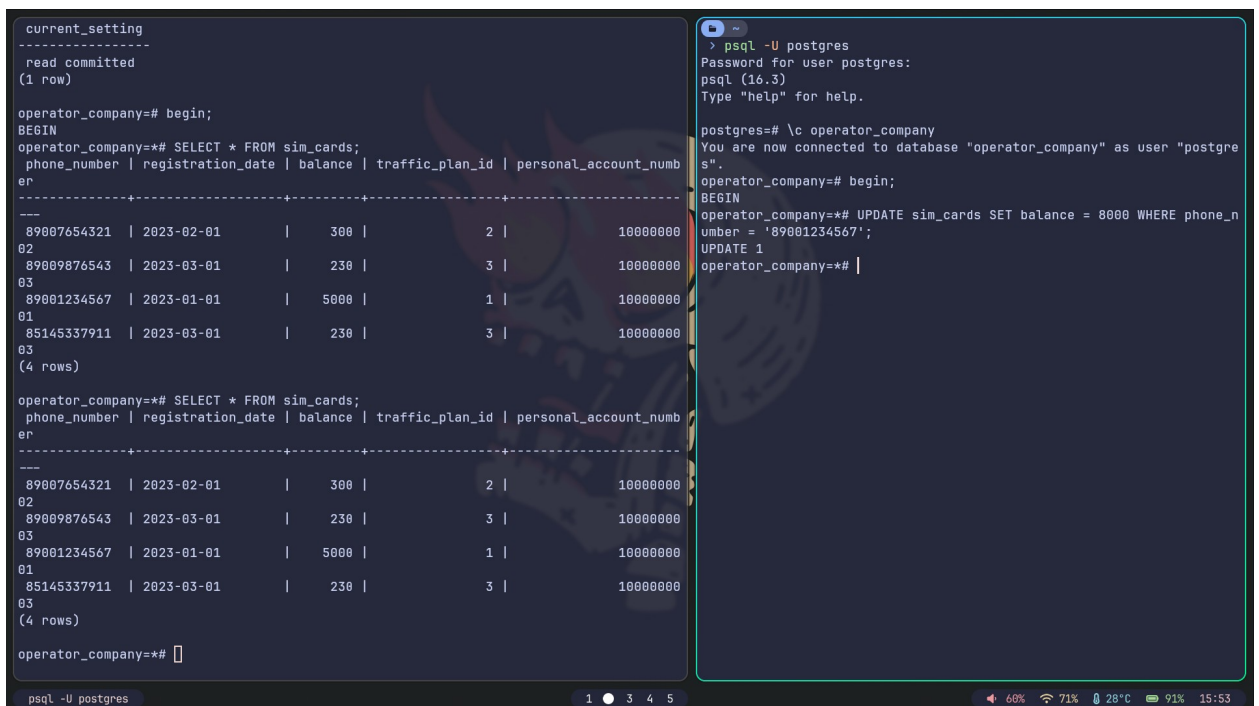
operator_company=# begin;
BEGIN
operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----+-----+-----+-----+-----
 89007654321 | 2023-02-01       | 300    | 2              | 10000000
 89009876543 | 2023-03-01       | 230    | 3              | 10000000
 89001234567 | 2023-01-01       | 5000   | 1              | 10000000
 85145337911 | 2023-03-01       | 230    | 3              | 10000000
(4 rows)

operator_company=#
```

```
> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=#
```

Figure 1: Просмотр в 1 транзакции до изменений во 2



```
current_setting
-----
read committed
(1 row)

operator_company=# begin;
BEGIN
operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----+-----+-----+-----+-----
 89007654321 | 2023-02-01       | 300    | 2              | 10000000
 89009876543 | 2023-03-01       | 230    | 3              | 10000000
 89001234567 | 2023-01-01       | 5000   | 1              | 10000000
 85145337911 | 2023-03-01       | 230    | 3              | 10000000
(4 rows)

operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----+-----+-----+-----+-----
 89007654321 | 2023-02-01       | 300    | 2              | 10000000
 89009876543 | 2023-03-01       | 230    | 3              | 10000000
 89001234567 | 2023-01-01       | 5000   | 1              | 10000000
 85145337911 | 2023-03-01       | 230    | 3              | 10000000
(4 rows)

operator_company=#
```

```
> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=# UPDATE sim_cards SET balance = 8000 WHERE phone_number = '89001234567';
UPDATE 1
operator_company=#
```

Figure 2: Изменение во 2 транзакции и просмотр в 1

```
03
89001234567 | 2023-01-01 | 5000 | 1 | 10000000
01
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
(4 rows)

operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 5000 | 1 | 10000000
01
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
(4 rows)

operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 8500 | 1 | 10000000
01
(4 rows)

operator_company=# |

> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=# UPDATE sim_cards SET balance = 8000 WHERE phone_number = '89001234567';
UPDATE 1
operator_company=# COMMIT;
COMMIT
operator_company=# |
```

Figure 3: Просмотр в 1 транзакции после комита во 2

REPEATABLE READ

```
> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SET
operator_company=# SELECT current_setting('transaction_isolation');
current_setting
-----
repeatable read
(1 row)

operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 8500 | 1 | 10000000
01
(4 rows)

operator_company=# |

> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=# |
```

Figure 4: Просмотр в 1 транзакции до изменения во 2

```
SET
operator_company=# SELECT current_setting('transaction_isolation');
current_setting
-----
repeatable read
(1 row)

operator_company=# SELECT * FROM sim_cards;
phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 8500 | 1 | 10000000
01
(4 rows)

operator_company=# SELECT * FROM sim_cards;
phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 8500 | 1 | 10000000
01
(4 rows)

operator_company=#
```

```
> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=# UPDATE sim_cards SET balance = 5000 WHERE phone_number = '89001234567';
UPDATE 1
operator_company=#
```

Figure 5: Изменение во 2 транзакции и просмотр в 1

```
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 8500 | 1 | 10000000
01
(4 rows)

operator_company=# SELECT * FROM sim_cards;
phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 8500 | 1 | 10000000
01
(4 rows)

operator_company=# SELECT * FROM sim_cards;
phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 8500 | 1 | 10000000
01
(4 rows)

operator_company=#
```

```
> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=# UPDATE sim_cards SET balance = 5000 WHERE phone_number = '89001234567';
UPDATE 1
operator_company=# COMMIT;
COMMIT
operator_company=#
```

Figure 6: Просмотр в 1 транзакции после комита во 2


```
01
(4 rows)

operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 8500 | 1 | 10000000
01
(4 rows)

operator_company=# COMMIT;
COMMIT
operator_company=# begin;
BEGIN
operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 5000 | 1 | 10000000
01
(4 rows)

operator_company=# |

> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=# UPDATE sim_cards SET balance = 5000 WHERE phone_number = '89001234567';
UPDATE 1
operator_company=# COMMIT;
COMMIT
operator_company=# |
```

Figure 7: Создание новой транзакции (после 1) и просмотр

READ UNCOMMITTED

```
> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=# SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET
operator_company=# SELECT current_setting('transaction_isolation');
 current_setting
-----
read uncommitted
(1 row)

operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 5000 | 1 | 10000000
01
(4 rows)

operator_company=# |

> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=# |
```

Figure 8: Просмотр в 1 транзакции до изменения во 2

```
SET
operator_company=# SELECT current_setting('transaction_isolation');
current_setting
-----
read uncommitted
(1 row)

operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----+-----+-----+-----+-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 5000 | 1 | 10000000
01
(4 rows)

operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----+-----+-----+-----+-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 5000 | 1 | 10000000
01
(4 rows)

operator_company=#
```

```
> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=# UPDATE sim_cards SET balance = 8300 WHERE phone_number = '89001234567';
UPDATE 1
operator_company=#
```

Figure 9: Изменение во 2 транзакции и просмотр в 1

```
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 5000 | 1 | 10000000
01
(4 rows)

operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----+-----+-----+-----+-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 5000 | 1 | 10000000
01
(4 rows)

operator_company=# SELECT * FROM sim_cards;
 phone_number | registration_date | balance | traffic_plan_id | personal_account_number
-----+-----+-----+-----+-----
89007654321 | 2023-02-01 | 300 | 2 | 10000000
02
89009876543 | 2023-03-01 | 230 | 3 | 10000000
03
85145337911 | 2023-03-01 | 230 | 3 | 10000000
03
89001234567 | 2023-01-01 | 8800 | 1 | 10000000
01
(4 rows)

operator_company=#
```

```
> psql -U postgres
Password for user postgres:
psql (16.3)
Type "help" for help.

postgres=# \c operator_company
You are now connected to database "operator_company" as user "postgres".
operator_company=# begin;
BEGIN
operator_company=# UPDATE sim_cards SET balance = 8300 WHERE phone_number = '89001234567';
UPDATE 1
operator_company=# COMMIT;
COMMIT
operator_company=#
```

Figure 10: Просмотр в 1 транзакции после комита во 2

Вывод: в ходе лабораторной работы были рассмотрены различные уровни изоляций в транзакции.



Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИУК «Информатика и управление»**

КАФЕДРА **ИУК4 «Программное обеспечение ЭВМ и информационные технологии»**

ЛАБОРАТОРНАЯ РАБОТА 8

**ДИСЦИПЛИНА: «ПРОЕКТИРОВАНИЕ СИСТЕМ ХРАНЕНИЯ И
ОБРАБОТКИ ДАННЫХ»**

Выполнил: студент гр. ИУК4-52Б _____ (____ Губин Е.В.____)
(Подпись) (Ф.И.О.)

Проверил: _____ (____ Глебов С.А.____)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:

- Оценка:

Калуга, 2024

Индексы

Индексы в SQL используются для оптимизации скорости выполнения запросов в базе данных. Они ускоряют доступ к данным, хранящимся в таблице, за счет упорядочивания и быстрого поиска значений в столбцах. Как правило создаются на поля таблиц, которые часто фигурируют в качестве параметров запроса.

Запрос без индексов:

```
select email from customers c
where personal_account_number = 1000000001;
```

Рис. 1: Пример запроса

Данный запрос без индекса происходил 0,123 секунды.

Создание индексов:

```
create index idx_customers_personal_account_number on customers (personal_account_number);

create unique index idx_customers_email on customers (email);

create index idx_traffic_plans_id on traffic_plans (traffic_plan_id);

create index idx_sim_cards_phone_number on sim_cards (phone_number);

create index idx_customer_to_customer on customer_to_customer (
    initiator_phone_number,
    interlocutor_phone_number
);

create index idx_customer_to_other on customer_to_other (
    other_phone_number,
    customer_phone_number
);

create index idx_customer_to_other_iniciator on customer_to_other (
    customer_is_initiator,
    other_phone_number,
    customer_phone_number
);

create index idx_messages_id on messages (message_id);

create index idx_messages_date on messages (message_date_time);

create index idx_calls_id on calls (call_id);

create index idx_calls_date on calls (call_date_time);
```

Рис. 2: Создание индексов

Запрос с индексом:

Выполним запрос, который выполнялся выше, но теперь с индексом:



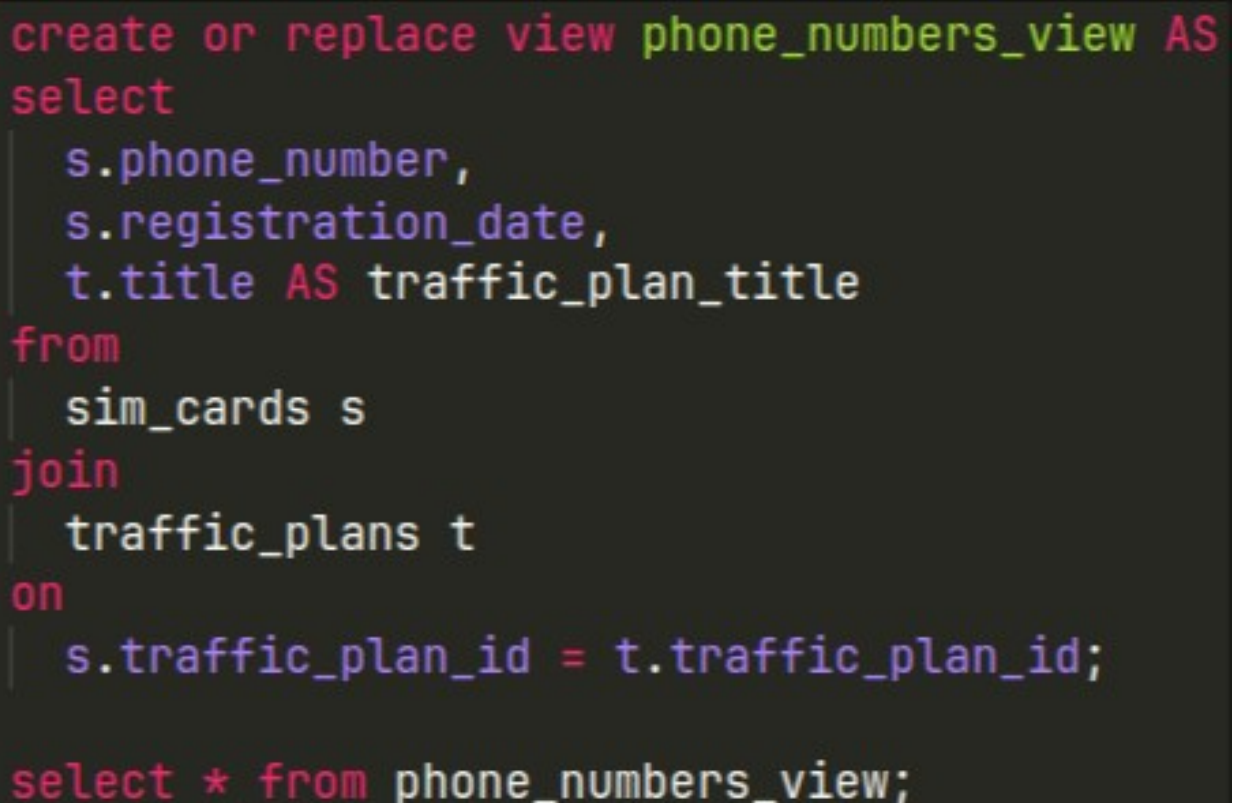
```
1 row(s) fetched - 0.001s, on 2024-12-08 at 15:49:36
```

Рис. 3: Результат выполнения запроса с индексом

Представления

View (представление) в SQL — это виртуальная таблица, которая создаётся на основе результата выполнения запроса. Представления не хранят данные сами по себе, а выступают как абстракция над базовыми таблицами. Они помогают организовать доступ к данным, упрощать сложные запросы и управлять безопасностью.

Создание представления:



```
create or replace view phone_numbers_view AS
select
  s.phone_number,
  s.registration_date,
  t.title AS traffic_plan_title
from
  sim_cards s
join
  traffic_plans t
on
  s.traffic_plan_id = t.traffic_plan_id;

select * from phone_numbers_view;
```

Рис. 4: Создание представления



```
select * from phone_numbers_view;
```

Рис. 5: Запрос представления

	A-Z phone_number	registration_date	A-Z traffic_plan_title
1	89007654321	2023-02-01	Стандартный
2	89009876543	2023-03-01	Премиум
3	85145337911	2023-03-01	Премиум
4	89001234567	2023-01-01	Базовый

Рис. 6: Результат запроса представления

Вывод: в ходе лабораторной работы были получены навыки по созданию и использованию индексов и представлений.



Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК «Информатика и управление»

КАФЕДРА ИУК4 «Программное обеспечение ЭВМ и информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА 9

**ДИСЦИПЛИНА: «ПРОЕКТИРОВАНИЕ СИСТЕМ ХРАНЕНИЯ И
ОБРАБОТКИ ДАННЫХ»**

Выполнил: студент гр. ИУК4-52Б _____ (____ Губин Е.В.____)
(Подпись) (Ф.И.О.)

Проверил: _____ (____ Глебов С.А.____)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:

- Оценка:

Калуга , 2024

В качестве реализации CRUD операций была взята курсовая работа. Операции созданы для работы с постами.

Сервер

Роутер для постов:

```
import { Router } from "express";
import postController from "../controllers/post-controller";
import authMiddleware from "../middlewares/auth-middleware";
import { param } from "express-validator";

const postRouter = Router();

postRouter.post("/", authMiddleware, postController.newPost);
postRouter.get("/all", authMiddleware, postController.getAllPosts);
postRouter.get(
  "/repost/:childrenPostId",
  param("childrenPostId").isNumeric(),
  authMiddleware,
  postController.getPostsByChildrenPostId
);
postRouter.get(
  "/posts/:userId",
  param("userId").isNumeric(),
  authMiddleware,
  postController.getPostsByUserId
);
postRouter.get(
  "/:postId",
  param("postId").isNumeric(),
  authMiddleware,
  postController.getPostById
);
postRouter.delete(
  "/:postId",
  param("postId").isNumeric(),
  authMiddleware,
  postController.deletePost
);

export default postRouter;
```

Контроллер для постов:

```
import { NextFunction, Request, Response } from "express";
import INewPost from "../interfaces/INewPost";
import postService from "../services/post-service";
import { validationResult } from "express-validator";
import ApiError from "../exceptions/ApiError";

class PostController {
```



```

    async newPost(req: Request, res: Response, next: NextFunction) {
      try {
        const newPost: INewPost = req.body;
        const authorId = Number(res.locals.userData.userId);
        const newPostData = await postService.newPost(newPost, authorId);
        res.json({ ...newPostData });
      } catch (error) {
        next(error);
      }
    }

    async getPostsByUserId(req: Request, res: Response, next: NextFunction) {
      try {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
          throw ApiError.BadRequest("Incorrect userId",
errors.array());
        }
        const userId = Number(req.params.userId);
        const posts = await postService.getPostsByUserId(userId);
        res.json({ posts });
      } catch (error) {
        next(error);
      }
    }

    async getPostById(req: Request, res: Response, next: NextFunction) {
      try {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
          throw ApiError.BadRequest("Incorrect post id",
errors.array());
        }
        const postId = Number(req.params.postId);
        const postData = await postService.getPostById(postId);
        res.json({ ...postData });
      } catch (error) {
        next(error);
      }
    }

    async deletePost(req: Request, res: Response, next: NextFunction) {
      try {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
          throw ApiError.BadRequest("Incorrect postId",
errors.array());
        }
        const postId = Number(req.params.postId);
        const post = await postService.deletePostById(postId);
        res.json({ ...post });
      } catch (error) {
        next(error);
      }
    }
  }

```

```

    async getPostsByChildrenPostId(
      req: Request,
      res: Response,
      next: NextFunction
    ) {
      try {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
          throw ApiError.BadRequest(
            "Incorrect children post id",
            errors.array()
          );
        }
        const childrenPostId = Number(req.params.childrenPostId);
        const posts = await postService.getPostsByChildrenPostId(
          childrenPostId
        );
        res.json({ posts });
      } catch (error) {
        next(error);
      }
    }
  }
}

async getAllPosts(req: Request, res: Response, next: NextFunction) {
  try {
    const posts = await postService.getAllPosts();
    res.json({ posts });
  } catch (error) {
    next(error);
  }
}

export default new PostController();

```

Сервис для постов:

```

import INewPost from "../interfaces/INewPost";
import db from "../db";
import IPostFromDataBase from "../interfaces/IPostFromDataBase";
import PostDto from "../dtos/post-dto";
import userService from "../user-service";
import ApiError from "../exceptions/ApiError";
import dateTimeService from "../dateTime-service";

class PostService {
  async newPost(newPost: INewPost, authorId: number) {
    if (!(await userService.userExistsById(authorId))) {
      throw ApiError.BadRequest("User with this id aren't exist");
    }
    if (
      newPost.childrenPostId &&
      !(await this.postExistsById(newPost.childrenPostId))
    )

```

```

    ) {
        throw ApiError.BadRequest("Children post isn't found");
    }

    const nowFormattedDateTime = dateTimeService.getNowDate();

    const postFromDataBase: IPostFromDataBase = (
        await db.query(
            `INSERT INTO posts (content, publication_date_time,
children_post_id, post_author_id) VALUES ($1, $2, $3, $4) RETURNING *`,
            [
                newPost.content,
                nowFormattedDateTime,
                newPost.childrenPostId,
                authorId,
            ]
        )
    ).rows[0];

    postFromDataBase.publication_date_time =
dateTimeService.formatDateTime(
    postFromDataBase.publication_date_time
);

    return new PostDto(postFromDataBase);
}

async getPostsByUserId(userId: number) {
    if (!(await userService.userIsExistsById(userId))) {
        throw ApiError.BadRequest("User with this id aren't exist");
    }
    if (await userService.userIsExistsById(userId)) {
        const posts: IPostFromDataBase[] = (
            await db.query(
                `SELECT * FROM posts WHERE post_author_id = $1`,
                [userId]
            )
        ).rows;
        return posts.map((post) => {
            post.publication_date_time = dateTimeService.formatDateTime(
                post.publication_date_time
            );
            return new PostDto(post);
        });
    }
    throw ApiError.BadRequest("User with this id aren't exists");
}

async getPostById(postId: number) {
    if (!(await this.postIsExistsById(postId))) {
        throw ApiError.ResourceNotFound();
    }
    const postData: IPostFromDataBase = (
        await db.query("SELECT * FROM posts WHERE post_id = $1",
[postId])

```

```

        ).rows[0];
        postData.publication_date_time = dateTimeService.formatDateTime(
            postData.publication_date_time
        );
        return new PostDto(postData);
    }

    async deletePostById(postId: number) {
        if (await this.postIsExistsById(postId)) {
            const post: IPostFromDataBase = (
                await db.query(
                    "DELETE FROM posts WHERE post_id = $1 RETURNING *",
                    [postId]
                )
            ).rows[0];
            post.publication_date_time = dateTimeService.formatDateTime(
                post.publication_date_time
            );
            return new PostDto(post);
        }
        throw ApiError.BadRequest("Post with this id aren't found");
    }

    async postIsExistsById(postId: number) {
        const postById: IPostFromDataBase[] = (
            await db.query(`SELECT * FROM posts WHERE post_id = $1`,
                [postId])
        ).rows;
        if (postById.length === 0) {
            return false;
        }
        return true;
    }

    async getPostsByChildrenPostId(childrenPostId: number) {
        if (!(await this.postIsExistsById(childrenPostId))) {
            throw ApiError.BadRequest("Children post isn't found");
        }
        const posts: IPostFromDataBase[] = (
            await db.query("SELECT * FROM posts WHERE children_post_id = $1",
                [
                    childrenPostId,
                ])
        ).rows;
        return posts.map((post) => {
            post.publication_date_time = dateTimeService.formatDateTime(
                post.publication_date_time
            );
            return new PostDto(post);
        });
    }

    async getPostIdsByAuthorId(authorId: number) {
        if (!(await userService.userIsExistsById(authorId))) {
            throw ApiError.BadRequest("Author id isn't found");
        }
    }

```

```

    }
    const ids: number[] = (
      await db.query(
        "SELECT post_id FROM posts WHERE post_author_id = $1",
        [authorId]
      )
    ).rows;
    return ids;
  }

  async getAllPosts() {
    const posts: IPostFromDataBase[] = (
      await db.query("SELECT * FROM posts", [])
    ).rows;
    return posts.map((post) => {
      post.publication_date_time = dateTimeService.formatDateTime(
        post.publication_date_time
      );
      return new PostDto(post);
    });
  }
}

export default new PostService();

```

Клиент

Axios для работы с сервером над постами:

```

import $api from "../http";
import IGetPost from "../interfaces/IResponses/IGetPost";

export default class PostService {
  static async getPostsByUserId(userId: number) {
    return await $api.get<{ posts: IGetPost[] }>(`/post/posts/${userId}`);
  }

  static async getPostById(postId: number) {
    return await $api.get<IGetPost>(`/post/${postId}`);
  }

  static async getPostsByChildrenPostId(childrenPostId: number) {
    return await $api.get<{ posts: IGetPost[] }>(
      `/post/repost/${childrenPostId}`
    );
  }

  static async newPost(
    content: string,
    childrenPostId: number | null = null
  ) {

```

```

        return await $api.post<IGetPost>("/post", { content,
childrenPostId });
    }

    static async deletePost(postId: number) {
        return await $api.delete<IGetPost>(`/post/${postId}`);
    }

    static async getAllPosts() {
        return await $api.get<{ posts: IGetPost[] }>("/post/all");
    }
}

```

Пользовательский интерфейс

Компонент пост:

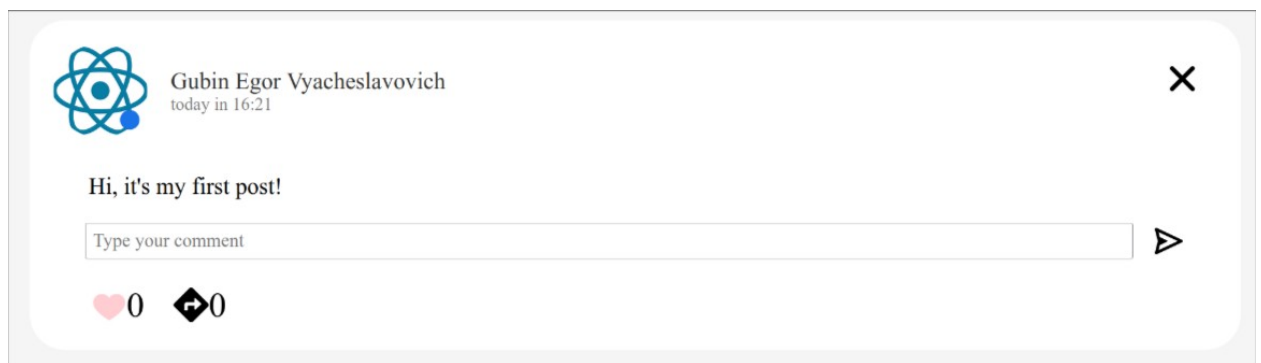


Рис. 1: Пост

```

import React, { ChangeEvent, useContext, useEffect, useState } from "react";
import s from "./Post.module.css";
import IPost from "../../interfaces/IProps/IPost";
import DateTimeService from "../../services/dateTime-service";
import PostService from "../../services/post-service";
import IGetPost from "../../interfaces/IResponses/IGetPost";
import IUser from "../../interfaces/IResponses/IUser";
import UserService from "../../services/user-service";
import ProfileImageService from "../../services/profileImage-service";
import classNames from "classnames";
import IGetComment from "../../interfaces/IResponses/IGetComment";
import Comment from "../../Comment/Comment";
import CommentService from "../../services/comment-service";
import { BiSend } from "react-icons/bi";
import { FcLikePlaceholder } from "react-icons/fc";
import { FcLike } from "react-icons/fc";
import ReactionService from "../../services/reaction-service";
import IGetReaction from "../../interfaces/IResponses/IGetReaction";
import { Context } from "../../";
import { FaDirections } from "react-icons/fa";
import { observer } from "mobx-react-lite";
import { IoClose } from "react-icons/io5";
import PostImageService from "../../services/postImage-service";

```

```

import ImageSlider from "../ImageSlider/ImageSlider";
import { useNavigate } from "react-router-dom";
import io from "socket.io-client";
import { globalSocket } from "../../globalSocket";
import { GrStatusGoodSmall } from "react-icons/gr";

const Post: React.FC<IPost> = ({
  post,
  isChild,
  setCreatePostFormIsOpened,
  setRepost,
  setPosts,
}) => {
  const { store } = useContext(Context);

  const [postAvatarImage, setPostAvatarImage] = useState("");
  const [author, setAuthor] = useState({} as IUser);
  const [postImages, setPostImages] = useState<string[]>([]);
  const [comments, setComments] = useState<IGetComment[]>([]);
  const [childPost, setChildPost] = useState<JSX.Element | null>(null);
  const [showAllComments, setShowAllComments] = useState<boolean>(false);
  const [newComment, setNewComment] = useState("");
  const [isReaction, setIsReaction] = useState(false);
  const [reactionsAmount, setReactionsAmount] = useState(0);
  const [repostsAmount, setRepostsAmount] = useState(0);
  const [isOnline, setIsOnline] = useState(false)

  const navigate = useNavigate();

  const openPageByAvatar = () => {
    if (store.user.userId !== post.postAuthorId) {
      navigate(`/profile/${post.postAuthorId}`);
    }
  };

  const loadPostAvatarImage = async () => {
    setPostAvatarImage(
      (await
ProfileImageService.getProfileImage(post.postAuthorId)).data
        .src
    );
  };

  const loadAuthor = async () => {
    setAuthor((await UserService.getUserById(post.postAuthorId)).data);
  };

  const loadPostImages = async () => {
    setPostImages(
      (await
PostImageService.getPostImages(post.postId)).data.postImages
    );
  };

  const loadComments = async () => {

```

```

    setComments(
      (
        await CommentService.getCommentsByPostId(post.postId)
      ).data.comments.sort((first, second) => {
        const dateFirst = new Date(first.commentDateTime);
        const dateSecond = new Date(second.commentDateTime);
        return dateSecond.getTime() - dateFirst.getTime();
      })
    );
  };

  const loadChildPost = async () => {
    if (post.childrenPostId) {
      const childPostData = (
        await PostService.getPostById(post.childrenPostId)
      ).data;
      setChildPost(<Post post={childPostData} isChild={true} />);
    }
  };

  const loadReactions = async () => {
    const reactionsData = (
      await ReactionService.getReactionsByPostId(post.postId)
    ).data.reactions;
    setReactionsAmount(reactionsData.length);
    for (let i = 0; i < reactionsData.length; i++) {
      if (reactionsData[i].reactionAuthorId === store.user.userId) {
        setIsReaction(true);
        return;
      }
    }
  };

  const loadReposts = async () => {
    setRepostsAmount(
      (await
        PostService.getPostsByChildrenPostId(post.postId)).data.posts
        .length
    );
  };

  const loadIsOnline = async () => {
    setIsOnline((await
      UserService.getStatus(post.postAuthorId)).data.isOnline)
  }

  useEffect(() => {
    loadAuthor();
    loadPostImages();
    loadChildPost();
    loadPostAvatarImage();
    loadReactions();
    loadReposts();
    loadIsOnline()
    if (!isChild) {

```



```

        loadComments();
    }
    const socket = io(globalSocket);
    socket.emit("subscribe_image", {
        userId: post.postAuthorId,
    });
    socket.emit("subscribe_like", {
        postId: post.postId,
        authorId: store.user.userId,
    });
    socket.emit("subscribe_online", {userId: post.postAuthorId });
    socket.on("set_image", () => {
        loadPostAvatarImage();
    });
    socket.on("set_like", ({ operation }: { operation: number }) => {
        console.log(operation)
        setReactionsAmount((prev) => prev + operation);
    });
    socket.on("set_status", ({ isOnline }: { isOnline: boolean }) => {
        setIsOnline(isOnline);
    });
    return () => {
        socket.off("set_status");
        socket.off("set_like");
        socket.off("set_image");
        socket.disconnect();
    };
}, []);

return (
    <div className={classNames(s.post, { [s.left_border]: isChild })}>
        <div className={s.post_header}>
            <div
                className={s.profile_post_image}
                onClick={openPageByAvatar}
                style={{
                    backgroundImage: `url(${postAvatarImage})`,
                }}
            >
                {isOnline && (
                    <GrStatusGoodSmall className={s.online} />
                )}</div>
            <div className={s.date_time_fio}>
                <div className={s.post_fio}>
                    {[
                        author.lastName,
                        author.firstName,
                        author.patronymic,
                    ].join(" ")}
                </div>
                <div className={s.pub_post_date_time}>
                    {DateTimeService.formDate(post.publicationDateTime)}
                </div>
            </div>
        </div>
    </div>

```

```

<div className={s.post_content}>{post.content}</div>
<ImageSlider images={postImages} />
{childPost}
{comments.length !== 0 &&
  comments.map((comment, index) => {
    if (showAllComments || comments.length ≤ 2)
      return (
        <Comment
          isMyPost={
            store.user.userId === post.postAuthorId
          }
          comment={comment}
          key={comment.commentId}
          setComments={setComments}
        />
      );
    if (index ≤ 1)
      return (
        <Comment
          isMyPost={
            store.user.userId === post.postAuthorId
          }
          comment={comment}
          key={comment.commentId}
          setComments={setComments}
        />
      );
    return null;
  })}
{comments.length > 2 ? (
  showAllComments ? (
    <span
      className={s.manage_comments}
      onClick={() => setShowAllComments(false)}
    >
      Hide comments
    </span>
  ) : (
    <span
      className={s.manage_comments}
      onClick={() => setShowAllComments(true)}
    >
      Show all commets...
    </span>
  )
) : null}
{!isChild ? (
  <div className={s.new_comment}>
    <textarea
      placeholder="Type your comment"
      className={s.comment_area}
      rows={1}
      value={newComment}
      onChange={(event) => {
        const textarea = event.target;

```

```

        textarea.style.height = "auto";

        textarea.style.height =
            textarea.scrollHeight + "px";
        setNewComment(event.target.value);
    }}
    ></textarea>
    <BiSend
        onClick={() => {
            CommentService.newComment(newComment,
post.postId)
                .then((response) => response.data)
                .then((data) =>
                    setComments((prev) => [data, ...prev])
                );
            setNewComment("");
        }}
        className={s.send_comment}
    />
</div>
) : null}
{isChild ? null : (
    <div className={s.reaction_repost}>
        <div
            className={s.reactions_amount}
            onClick={
                isReaction
                    ? () => {
                        ReactionService.deleteReaction(
                            post.postId
                        )
                            .then((response) => response.data)
                            .then((data) => {
                                setReactionsAmount(
                                    (prev) => prev - 1
                                );
                                setIsReaction(false);
                            });
                        const socket = io(globalSocket);
                        socket.emit("change_like", {
                            postId: post.postId,
                            operation: -1,
                            authorId: store.user.userId,
                        });
                    }
                    : () => {
                        ReactionService.newReaction(post.postId)
                            .then((response) => response.data)
                            .then((data) => {
                                setReactionsAmount(
                                    (prev) => prev + 1
                                );
                                setIsReaction(true);

```

```

        });
        const socket = io(globalSocket);
        socket.emit("change_like", {
            postId: post.postId,
            operation: 1,
            authorId: store.user.userId,
        });
    }
}
>
{isReaction ? (
    <FcLike className={s.reaction_button} />
) : (
    <FcLikePlaceholder
className={s.reaction_button} />
    )}{ " "}
    {reactionsAmount}{ " "}
</div>
<div
    className={classNames({
        [s.reposts_amount]:
            post.postAuthorId === store.user.userId,
        [s.repost_amount_unself]:
            post.postAuthorId === store.user.userId,
    })}
    onClick={
        post.postAuthorId === store.user.userId
        ? () => {}
        : () => {
            if (
                setCreatePostFormIsOpened &&
                setRepost
            ) {
                setCreatePostFormIsOpened(true);
                setRepost(post);
            }
        }
    }
}
>
    <FaDirections className={s.repost_button} />{ " "}
    {repostsAmount}
</div>
</div>
)}
{isChild || post.postAuthorId === store.user.userId ? null : (
    <IoClose
        className={s.close}
        onClick={() => {
            PostService.deletePost(post.postId)
                .then((response) => response.data)
                .then((data) => {
                    if (setPosts)
                        setPosts((prev) =>
                            prev.filter(
                                (postData) =>

```

```

post.postId
                                postData.postId !==
                                )
                                );
                                const socket = io(globalSocket);
                                socket.emit("delete_post", {
                                    postId: data.postId,
                                    authorId: data.postAuthorId,
                                });
                                });
                                }
                                />
                                })
                                </div>
                                );
                                };

export default observer(Post);

```

Форма для создания поста:

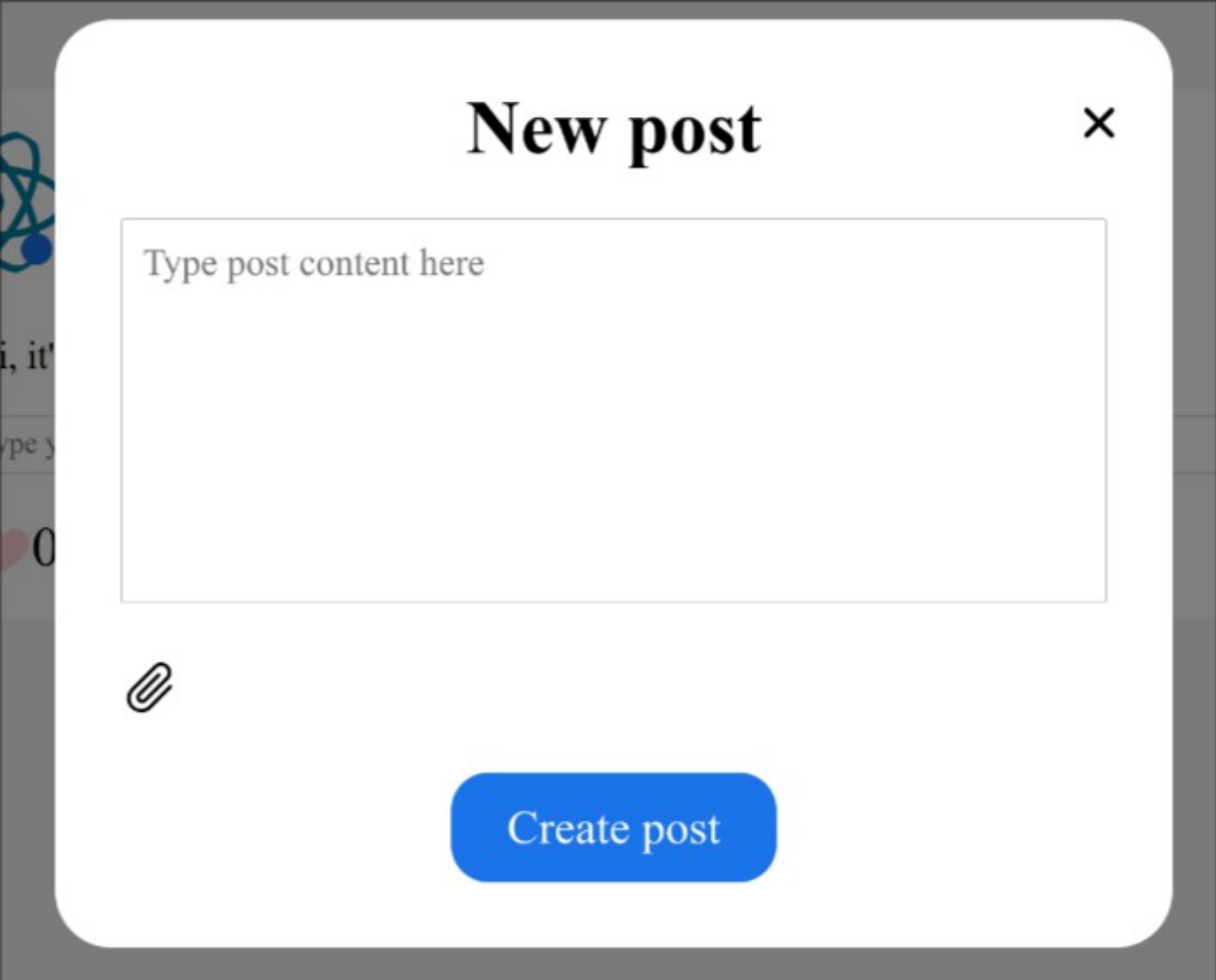


Рис. 2: Форма для создания поста

```
import React, { ChangeEvent, useRef, useState } from "react";
```

```

import s from "./NewPostForm.module.css";
import { FormButton } from "../UI/FormButton/FormButton";
import { GoPaperclip } from "react-icons/go";
import PostService from "../../services/post-service";
import PostImageService from "../../services/postImage-service";
import INewPostForm from "../../interfaces/IProps/INewPostForm";
import io from "socket.io-client";
import { globalSocket } from "../../globalSocket";

export const NewPostForm: React.FC<INewPostForm> = ({
  setPosts,
  repost,
  setRepost,
  setCreatePostFormIsOpened,
  isFromPostsPage,
}) => {
  const [content, setContent] = useState("");
  const fileInputRef = useRef<HTMLInputElement>(null);
  const [files, setFiles] = useState<File[]>([]);
  const [filesImages, setFilesImages] = useState<string[]>([]);

  const setImages = (event: ChangeEvent<HTMLInputElement>) => {
    const filesEvent = event.target.files;
    if (!filesEvent || filesEvent.length === 0) {
      return;
    }
    const permittedTypes = ["image/jpeg", "image/jpg", "image/png"];
    for (let i = 0; i < filesEvent.length; i++) {
      if (!permittedTypes.includes(filesEvent[i].type)) {
        return;
      }
    }
    const filesArray = Array.from(filesEvent);
    setFiles(filesArray);
    setFilesImages(
      filesArray.map(
        (fileData) => `url(${URL.createObjectURL(fileData)})`
      )
    );
  };

  return (
    <div className={s.new_post_form}>
      <div className={s.container}>
        <textarea
          rows={8}
          placeholder="Type post content here"
          className={s.content}
          value={content}
          onChange={(event) => setContent(event.target.value)}
        ></textarea>
        <div className={s.selected_images}>
          <GoPaperclip
            className={s.clip}
            onClick={() => fileInputRef.current?.click()}
          />
        </div>
      </div>
    </div>
  );
};

```

```

        />
        {filesImages.map((file) => (
            <div
                className={s.image}
                style={{
                    backgroundImage: file,
                }}
            ></div>
        ))}
    </div>
    <input
        ref={fileInputRef}
        type="file"
        multiple={true}
        accept=".jpg, .jpeg, .png"
        onChange={setImages}
        className={s.file_input}
    />
</div>
<FormButton
    onClick={
        repost
        ? async () => {
            const post = (
                await PostService.newPost(
                    content,
                    repost.postId
                )
            ).data;
            await PostImageService.newPostImages(
                files,
                post.postId
            );
            setRepost(null);
            setContent("");
            setFiles([]);
            setFilesImages([]);
            setCreatePostFormIsOpened(false);
            if (isFromPostsPage)
                setPosts((prev) => [post, ...prev]);
            const socket = io(globalSocket);
            socket.emit("new_post", { post });
        }
        : async () => {
            const post = (await
PostService.newPost(content))
                .data;
            await PostImageService.newPostImages(
                files,
                post.postId
            );
            setContent("");
            setFiles([]);
            setFilesImages([]);
            setCreatePostFormIsOpened(false);

```

```

        setPosts((prev) => [post, ...prev]);
        const socket = io(globalSocket);
        socket.emit("new_post", { post });
    }
}
type="button"
>
    Create post
</FormButton>
</div>
);
};

```

Страница постов:

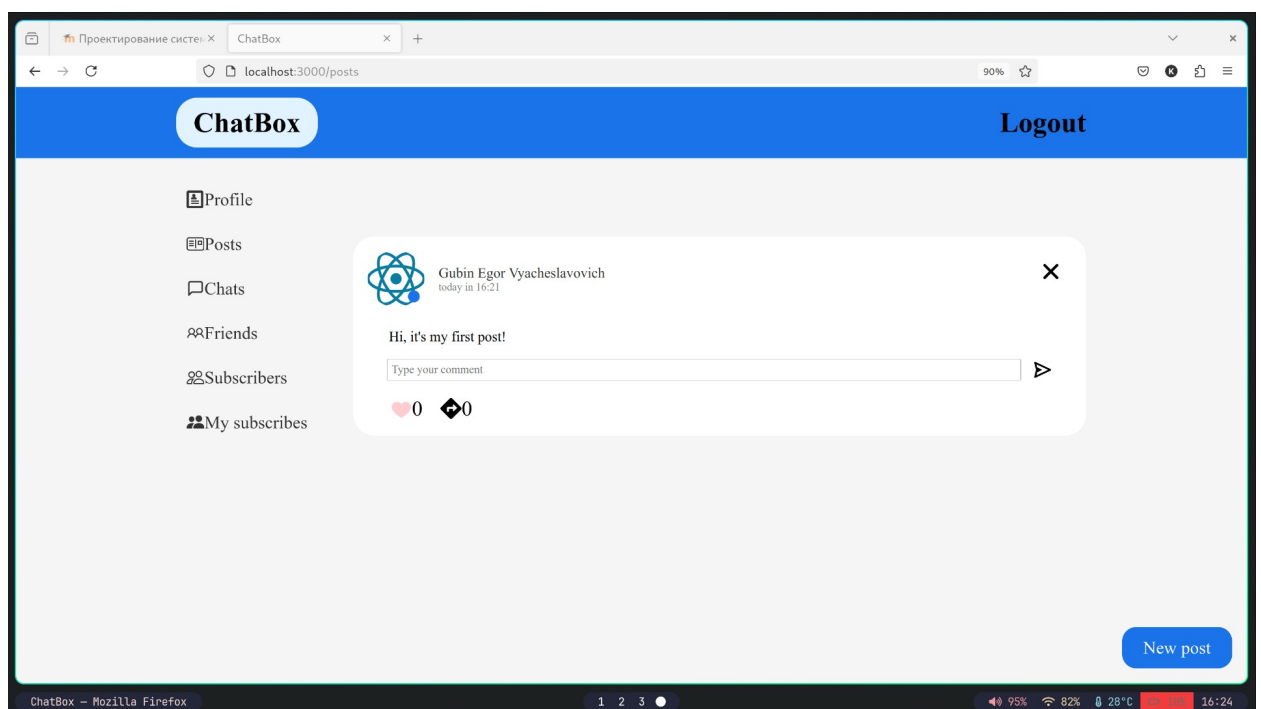


Рис. 3: Страница постов

```

import React, { useEffect, useState } from "react";
import s from "./PostsPage.module.css";
import IGetPost from "../../interfaces/IResponses/IGetPost";
import PostService from "../../services/post-service";
import Post from "../Post/Post";
import { ModalWindow } from "../../ModalWindow/ModalWindow";
import { NewPostForm } from "../../NewPostForm/NewPostForm";
import { FormButton } from "../../UI/FormButton/FormButton";

export const PostsPage: React.FC = () => {
    const [posts, setPosts] = useState<IGetPost[]>([]);
    const [createPostFormIsOpened, setCreatePostFormIsOpened] =
    useState(false);
    const [repost, setRepost] = useState<IGetPost | null>(null);

    const loadPosts = async () => {

```



```

const postData = (await PostService.getAllPosts()).data.posts;
setPosts(
  postData.sort((first, second) => {
    const dateFirst = new Date(first.publicationDateTime);
    const dateSecond = new Date(second.publicationDateTime);
    return dateSecond.getTime() - dateFirst.getTime();
  })
);
};

useEffect(() => {
  loadPosts();
}, []);

return (
  <div>
    {posts.map((post) => (
      <Post
        key={post.postId}
        post={post}
        isChild={false}
        setRepost={setRepost}
        setCreatePostFormIsOpened={setCreatePostFormIsOpened}
        setPosts={setPosts}
      />
    ))}
    <FormButton
      className={s.new_post}
      type="button"
      onClick={() => setCreatePostFormIsOpened(true)}
    >
      New post
    </FormButton>
    <ModalWindow
      isOpened={createPostFormIsOpened}
      setIsOpened={setCreatePostFormIsOpened}
      header="New post"
    >
      <NewPostForm
        setCreatePostFormIsOpened={setCreatePostFormIsOpened}
        setPosts={setPosts}
        repost={repost}
        setRepost={setRepost}
        isFromPostsPage={true}
      />
    </ModalWindow>
  </div>
);
};

```

Вывод: в ходе лабораторной работы были реализованы CRUD операции для работы с постами.

