

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5 «Процедуры, функции,
триггеры в PostgreSQL» по дисциплине «Проектирование и
реализация баз данных»**

Обучающийся (Березина Софья Константиновна)

Факультет прикладной информатики

Группа K3239

Направление подготовки 09.03.03 Прикладная информатика

Образовательная программа Мобильные и сетевые технологии 2023 **Преподаватель**
Говорова Марина Михайловна

Санкт-Петербург
2024/2025

ОГЛАВЛЕНИЕ

1. ЦЕЛЬ РАБОТЫ	3
2. ПРАКТИЧЕСКОЕ ЗАДАНИЕ.....	4
3. ВЫПОЛНЕНИЕ РАБОТЫ	5
4. ВЫВОДЫ.....	13

1. ЦЕЛЬ РАБОТЫ

Цель работы: овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

2. ПРАКТИЧЕСКОЕ ЗАДАНИЕ

1. Создать 3 процедуры для индивидуальной БД согласно варианту (часть 4 ЛР 2). Допустимо использование IN/OUT параметров. Допустимо создать авторские процедуры. (3 балла)
2. Создать триггеры для индивидуальной БД согласно варианту: Вариант 2.1. 3 триггера - 3 балла (min). Допустимо использовать триггеры логирования из практического занятия по функциям и триггерам.

3. ВЫПОЛНЕНИЕ РАБОТЫ

3.1 Индивидуальное задание

Вариант 19. БД «Банк»

Описание предметной области: Система обеспечивает работу с вкладами кредитами клиентов банка.

Клиенты банка имеют вклады и кредиты различных видов. Для вкладов и кредитов может использоваться различная валюта.

Сотрудники банка заключают договоры с клиентами. Фиксируется сотрудник, заключивший договор.

Ежемесячно начисляется процент по вкладу, и полученная сумма добавляется к сумме вклада заказчика. Вкладчик имеет право снимать проценты по вкладу или всю сумму вклада с процентами по истечении срока вклада. При снятии денег до истечения срока вклада процент за текущий месяц не начисляется.

Кредит выдается на определенный срок. Формируется график выплат, который получает клиент при заключении договора, в котором ежемесячно указывается сумма выплаты по вкладу и сумма выплаты по процентам банку. Хранится информация по своевременности ежемесячных выплат.

БД должна содержать следующий минимальный набор сведений: ФИО сотрудника. Возраст сотрудника. Адрес сотрудника. № телефона сотрудник. Паспортные данные сотрудника. Должность сотрудника. Оклад сотрудника (зависит от должности). Наименование вклада. Описание вклада. Минимальный срок вклада. Минимальная сумма вклада. Процент по вкладу. Срок вклада. Процентная ставка. Код валюты. Наименование валюты. ФИО вкладчика. Адрес вкладчика. Телефон клиента. E-mail вкладчика. Паспортные данные. Номер договора. Дата вклада. Дата возврата. Сумма вклада. Сумма возврата. Данные по кредиту. Число выплаты ежемесячно (нельзя указывать 29, 30 и 31). Должность сотрудника. Количество ставок (по штатному расписанию). Дополните состав атрибутов на основе анализа предметной области.

Задание 4. Создать хранимые процедуры:

- о текущей сумме вклада и сумме начисленного за месяц процента для заданного клиента;
- добавить данные о новом вкладе клиента;
- найти клиентов банка, не имеющих задолженности по кредитам.

3.2 Наименование БД

Наименование базы данных: «bank»

3.3 Задание №1

Процедура 1: Текущая сумма вклада и сумма начисленного за месяц процента

```
CREATE OR REPLACE PROCEDURE bank_schema."расчет_вклада_и_процентов"(
  IN p_id_клиента INTEGER,
  OUT p_текущая_сумма NUMERIC(15,2),
  OUT p_начисленные_проценты NUMERIC(15,2)
)
AS $$
BEGIN
  IF NOT EXISTS (SELECT 1 FROM bank_schema."клиент" WHERE "id_клиента" =
p_id_клиента) THEN
```

```

        RAISE EXCEPTION 'Клиент с ID % не найден', p_id_клиента;
    END IF;

    SELECT
        SUM("сумма"),
        SUM("сумма" * "процентная_ставка" / 100 / 12)
    INTO
        p_текущая_сумма,
        p_начисленные_проценты
    FROM bank_schema."договор_вклада"
    WHERE "id_клиента" = p_id_клиента
    AND ("дата_возврата" IS NULL OR "дата_возврата" > CURRENT_DATE);

    IF p_текущая_сумма IS NULL THEN
        p_текущая_сумма := 0;
        p_начисленные_проценты := 0;
    END IF;
END;
$$ LANGUAGE plpgsql;

[bank=# CALL bank_schema."расчет_вклада_и_процентов"(4, NULL, NULL);
 p_текущая_сумма | p_начисленные_проценты
-----+-----
      100000.00 |   195.000000000000000000
(1 row)

```

Процедура 2: Добавление данных о новом вкладе клиента

```

CREATE OR REPLACE PROCEDURE bank_schema."добавить_вклад"(
    p_id_клиента INTEGER,
    p_id_вида_вклада INTEGER,
    p_сумма NUMERIC(10,2),
    p_код_валюты CHAR(3) DEFAULT 'RUB',
    p_дата_вклада DATE DEFAULT CURRENT_DATE,
    p_дата_возврата DATE DEFAULT NULL
)
AS $$
DECLARE
    v_минимальная_сумма NUMERIC(10,2);
    v_процентная_ставка NUMERIC(5,2);
    v_id_сотрудника INTEGER := 1;
    v_new_id INTEGER;
BEGIN
    IF NOT EXISTS (SELECT 1 FROM bank_schema."клиент" WHERE "id_клиента" =
p_id_клиента) THEN
        RAISE EXCEPTION 'Клиент с ID % не найден', p_id_клиента;
    END IF;

    SELECT
        "минимальная_сумма",
        "процентная_ставка"
    INTO

```

```

        v_минимальная_сумма,
        v_процентная_ставка
FROM bank_schema."вид_вклада"
WHERE "id_вида_вклада" = p_id_вида_вклада;

IF NOT FOUND THEN
    RAISE EXCEPTION 'Вид вклада с ID % не найден', p_id_вида_вклада;
END IF;

IF p_сумма < v_минимальная_сумма THEN
    RAISE EXCEPTION 'Минимальная сумма вклада: %. Текущая: %',
        v_минимальная_сумма, p_сумма;
END IF;

IF EXISTS (SELECT 1 FROM information_schema.columns
            WHERE table_schema = 'bank_schema'
            AND table_name = 'договор_вклада'
            AND column_name = 'id_договора_вклада'
            AND column_default IS NULL) THEN
    SELECT COALESCE(MAX("id_договора_вклада"), 0) + 1 INTO v_new_id
    FROM bank_schema."договор_вклада";
ELSE
    v_new_id := NULL;
END IF;

INSERT INTO bank_schema."договор_вклада" (
    "id_договора_вклада",
    "id_вида_вклада",
    "id_сотрудника",
    "id_клиента",
    "код_валюты",
    "дата_вклада",
    "дата_возврата",
    "сумма",
    "процентная_ставка"
) VALUES (
    v_new_id,
    p_id_вида_вклада,
    v_id_сотрудника,
    p_id_клиента,
    p_код_валюты,
    p_дата_вклада,
    p_дата_возврата,
    p_сумма,
    v_процентная_ставка
);
END;
$$ LANGUAGE plpgsql;

Вызовем процедуру:
CALL bank_schema."добавить_вклад"(
    p_id_клиента := 2,

```

```

p_id_вида_вклада := 5,
p_сумма := 50000.00,
p_код_валюты := 'USD',
p_дата_вклада := '2025-05-21',
p_дата_возврата := '2026-05-21'
);

```

До выполнения:

	id_договора_вклада [PK] integer	id_вида_вклада integer	id_сотрудника integer	id_клиента integer	код_валюты character (3)	дата_вклада date	дата_возврата date	сумма numeric (10,2)	процентная_ставка numeric (5,2)
1	5	1	2	2	RUB	2025-02-01	2029-02-01	50000.00	5.00
2	2	2	4	2	RUB	2024-02-15	2026-02-15	500000.00	6.45
3	3	3	1	4	RUB	2025-01-20	2028-03-01	100000.00	2.34
4	4	5	3	4	USD	2025-01-20	2025-04-30	10000.00	3.21
5	1	2	1	1	GBP	2024-01-10	2025-04-30	200000.00	8.00

После выполнения:

	id_договора_вклада [PK] integer	id_вида_вклада integer	id_сотрудника integer	id_клиента integer	код_валюты character (3)	дата_вклада date	дата_возврата date	сумма numeric (10,2)	процентная_ставка numeric (5,2)
1	5	1	2	2	RUB	2025-02-01	2029-02-01	50000.00	5.00
2	2	2	4	2	RUB	2024-02-15	2026-02-15	500000.00	6.45
3	3	3	1	4	RUB	2025-01-20	2028-03-01	100000.00	2.34
4	4	5	3	4	USD	2025-01-20	2025-04-30	10000.00	3.21
5	1	2	1	1	GBP	2024-01-10	2025-04-30	200000.00	8.00
6	6	5	1	2	USD	2025-05-21	2026-05-21	50000.00	4.50

Процедура 3: Найти клиентов банка, не имеющих задолженности по кредитам

```

CREATE OR REPLACE PROCEDURE bank_schema."клиенты_без_задолженности"()
AS $$
DECLARE
    v_record RECORD;
    v_cursor CURSOR FOR
        SELECT DISTINCT к."id_клиента", к."фιο", к."номер_телефона"
        FROM bank_schema."клиент" к
        WHERE NOT EXISTS (
            SELECT 1
            FROM bank_schema."договор_кредита" кр
            JOIN bank_schema."график_выплат" гр ON кр."id_договора_кредита" =
гр."id_договора"
            WHERE кр."id_клиента" = к."id_клиента"
            AND гр."статус_оплаты" = FALSE
            AND гр."дата_назначенной_выплаты" < CURRENT_DATE
        )
        ORDER BY к."фιο";
BEGIN
    RAISE NOTICE 'Список клиентов без задолженностей:';
    RAISE NOTICE '-----';

    OPEN v_cursor;
    LOOP
        FETCH v_cursor INTO v_record;
        EXIT WHEN NOT FOUND;

        RAISE NOTICE 'ID: %, ФИО: %',

```



```

        v_record."id_клиента",
        v_record."фio";
    END LOOP;
    CLOSE v_cursor;
END;
$$ LANGUAGE plpgsql;

|bank=# CALL bank_schema."клиенты_без_задолженности"();
NOTICE:  Список клиентов без задолженностей:
NOTICE:  -----
NOTICE:  ID: 2, ФИО: Борисова Ольга Игоревна
NOTICE:  ID: 4, ФИО: Григорьева Татьяна Викторовна
NOTICE:  ID: 5, ФИО: Дмитриев Сергей Александрович
NOTICE:  ID: 6, ФИО: Иванов Иван Иванович
CALL

```

3.4 Задание №2

Триггер 1: Логирование изменений в таблице клиентов

Создаем таблицу для логов:

```

CREATE TABLE IF NOT EXISTS bank_schema."логи_клиентов" (
    id_log SERIAL PRIMARY KEY,
    operation VARCHAR(10) NOT NULL,
    client_id INTEGER,
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    old_data JSONB,
    new_data JSONB
);

```

Создаем триггерную функцию:

```

CREATE OR REPLACE FUNCTION bank_schema."log_client_changes"()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO bank_schema."логи_клиентов" (operation, client_id, new_data)
        VALUES ('INSERT', NEW.id_клиента, to_jsonb(NEW));
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO bank_schema."логи_клиентов" (operation, client_id, old_data, new_data)
        VALUES ('UPDATE', NEW.id_клиента, to_jsonb(OLD), to_jsonb(NEW));
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO bank_schema."логи_клиентов" (operation, client_id, old_data)
        VALUES ('DELETE', OLD.id_клиента, to_jsonb(OLD));
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

```

Создаем триггер:

```

CREATE TRIGGER "client_changes_log"

```

AFTER INSERT OR UPDATE OR DELETE ON bank_schema."клиент"
FOR EACH ROW EXECUTE FUNCTION bank_schema."log_client_changes"());

Query

Query History

Scratch Pad

X

1

SELECT * FROM bank_schema."логи_клиентов" ORDER BY changed_at DESC LIMIT 3;

Data Output

Messages

Notifications

Showing rows: 1 to 2

Page No: 1

of 1

	id_log [PK] integer	operation character varying (10)	client_id integer	changed_at timestamp without time zone	old_data jsonb	new_data jsonb
1	1	INSERT	7	2025-05-21 12:05:29.229538	[null]	{ "email": "sidorov@mail.ru", "фio": "Сидоров Алексей Петрович", "адрес": "г. Москва, ул. Лени
2	2	INSERT	8	2025-05-21 12:05:29.229538	[null]	{ "email": "kuznetsova@mail.ru", "фio": "Кузнецова Анна Сергеевна", "адрес": "г. Санкт-Пете

Триггер 2: проверка платежей по кредитам

CREATE OR REPLACE FUNCTION bank_schema."validate_payment"
RETURNS TRIGGER AS \$\$
BEGIN

IF NEW.дата_фактической_выплаты IS NULL
AND NEW.дата_назначенной_выплаты < CURRENT_DATE THEN
NEW.статус_оплаты := FALSE;
ELSIF NEW.дата_фактической_выплаты IS NOT NULL THEN
NEW.статус_оплаты := TRUE;
END IF;

RETURN NEW;
END;
\$\$ LANGUAGE plpgsql;

CREATE TRIGGER "payment_validation"
BEFORE INSERT OR UPDATE ON bank_schema."график_выплат"
FOR EACH ROW EXECUTE FUNCTION bank_schema."validate_payment"());

Сделаем вставку:

INSERT INTO bank_schema."график_выплат"
(id_платежа, id_договора, дата_назначенной_выплаты, сумма_процентов,
сумма_основного_долга)
VALUES (6, 1, CURRENT_DATE - 1, 1000, 5000);

	id_платежа [PK] integer	id_договора integer	дата_назначенной_выплаты date	дата_фактической_выплаты date	сумма_процентов numeric (10,2)	сумма_основного_долга numeric (10,2)	статус_оплаты boolean
1	1	1	2025-02-15	2025-02-15	5208.33	13888.89	true
2	2	1	2025-03-15	2025-03-15	5069.44	13888.89	false
3	3	2	2024-07-01	2024-07-01	21875.00	12500.00	true
4	4	2	2024-08-01	2024-08-01	21831.60	12500.00	true
5	5	3	2025-04-10	2025-04-15	11562.50	25000.00	false
6	6	1	2025-05-20	[null]	1000.00	5000.00	false

Триггер 3: автоматическое обновление статуса кредита

```
ALTER TABLE bank_schema."договор_кредита"
ADD COLUMN IF NOT EXISTS "статус" VARCHAR(20) DEFAULT 'АКТИВЕН';
```

```
CREATE OR REPLACE FUNCTION bank_schema."update_credit_status"()
RETURNS TRIGGER AS $$
BEGIN
```

```
    IF NEW."статус_оплаты" = TRUE THEN
```

```
        PERFORM 1 FROM bank_schema."график_выплат"
        WHERE "id_договора" = NEW."id_договора"
        AND "статус_оплаты" = FALSE
        AND "дата_назначенной_выплаты" <= CURRENT_DATE;
```

```
    IF NOT FOUND THEN
```

```
        UPDATE bank_schema."договор_кредита"
        SET
            "статус" = 'ПОГАШЕН',
            "дата_погашения" = CURRENT_DATE
        WHERE "id_договора_кредита" = NEW."id_договора";
```

```
    END IF;
```

```
END IF;
```

```
    RETURN NEW;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```


```
CREATE TRIGGER "credit_status_trigger"
AFTER INSERT OR UPDATE OF "статус_оплаты" ON bank_schema."график_выплат"
FOR EACH ROW EXECUTE FUNCTION bank_schema."update_credit_status"();
```

Сделаем проверку и внесем новые данные:

```
INSERT INTO bank_schema."договор_кредита" (
    "id_сотрудника", "id_клиента", "код_валюты",
    "дата_выдачи", "сумма", "процентная_ставка",
    "число_платежа", "id_вида_кредита"
) VALUES (
    1, 1, 'RUB', '2025-01-01', 100000, 12, 3, 1
);
```



```
INSERT INTO bank_schema."график_выплат" VALUES
(7, 1, '2025-02-01', '2025-02-01', 1000, 30000, TRUE),
(8, 1, '2025-03-01', '2025-03-01', 800, 30000, TRUE),
(9, 1, '2025-04-01', NULL, 600, 30000, FALSE);
```

```
SELECT "статус" FROM bank_schema."договор_кредита" WHERE "id_договора_кредита" = 2;
```

	статус character varying (20) 
1	АКТИВЕН

```
UPDATE bank_schema."график_выплат"  
SET "статус_оплаты" = TRUE, "дата_фактической_выплаты" = CURRENT_DATE  
WHERE "id_платежа" = 9;
```

```
SELECT "статус", "дата_погашения" FROM bank_schema."договор_кредита" WHERE  
"id_договора_кредита" = 2;
```

	статус character varying (20) 	дата_погашения date 
1	ПОГАШЕН	2025-05-21

4. ВЫВОДЫ

В ходе выполнения лабораторной работы по теме "Процедуры, функции и триггеры в PostgreSQL" мною были успешно реализованы и протестированы различные объекты базы данных для банковской системы.

Главным выводом из выполненной работы стало понимание, что грамотное использование хранимых процедур и триггеров позволяет не только упростить прикладной код, но и значительно повысить надежность системы в целом. Автоматизированные проверки на уровне СУБД обеспечивают целостность данных даже в случае ошибок в прикладном коде.

В заключение можно сказать, что лабораторная работа дала ценный практический опыт работы с продвинутыми возможностями PostgreSQL. Полученные знания будут полезны при разработке реальных банковских систем, где требования к надежности и согласованности данных особенно высоки. Все поставленные задачи выполнены в полном объеме, что подтверждается успешными результатами тестирования.