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 Недиков Михаил Олегович

Факультет прикладной информатики

Группа K3239

Направление подготовки 09.03.03 Прикладная информатика

Образовательная программа Мобильные и сетевые технологии 2025

Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2025/2026

1. 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.

   Вариант 2.2. 7 оригинальных триггеров - 7 баллов (max).

3. Выполнение (19 Вариант “Банк”)

Задание 4. Создать хранимые процедуры:

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

3.1 Хранимые процедуры и функции

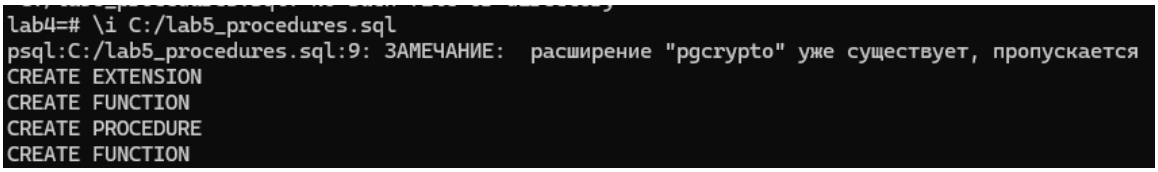
P1 – fn_deposit_info

Возвратить вклады клиента, актуальную сумму и процент за последний месяц

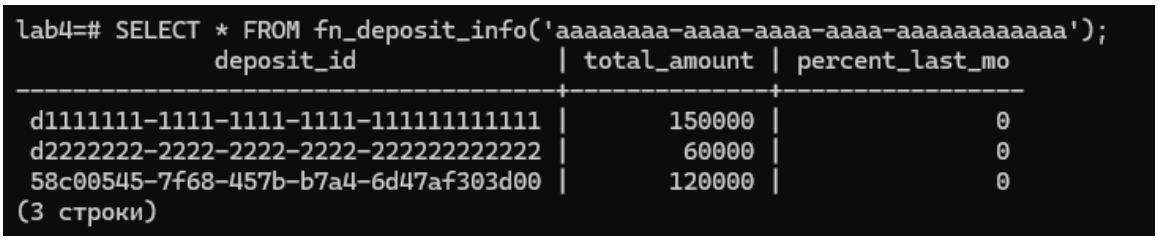
CREATE OR REPLACE FUNCTION fn_deposit_info(p_client uuid)

```
RETURNS TABLE (
  deposit_id       uuid,
  total_amount     numeric,
  percent_last_mo  numeric)
LANGUAGE plpgsql
AS $$
BEGIN
  RETURN QUERY
  SELECT dc."Deposit_Contract_ID",
        (dc."Deposit_Amount" + dc."Accrued_Interest")::numeric,  -- ← cast
        COALESCE((
          SELECT SUM(ps."Amount")
          FROM   "Deposit_Payment_Schedule" ps
          WHERE  ps."Deposit_Contract_ID" = dc."Deposit_Contract_ID"
            AND  ps."Date" >= date_trunc('month', current_date) - interval '1
month'
            AND  ps."Date" <  date_trunc('month', current_date)
        ), 0)::numeric
  FROM   "Deposit_Contract" dc
  WHERE  dc."Client_ID" = p_client;
END;
$$;
```

Скрин создания всех необходимых процедур и функций



Скрин вызова



P2 – sp_add_deposit

Добавить новый вклад клиенту

```
CREATE OR REPLACE PROCEDURE sp_add_deposit (
    p_deposit_data_id uuid,
    p_client_id       uuid,
    p_currency_id     uuid,
    p_term_months     int,
    p_amount          numeric,
    p_employee_id     uuid )
LANGUAGE plpgsql
AS $$
DECLARE v_id uuid := gen_random_uuid();
BEGIN
  INSERT INTO "Deposit_Contract"(
```

```
    "Deposit_Contract_ID","Deposit_ID","Client_ID","Opening_Date",
    "Deposit_Term","Deposit_Amount","Employee_ID","Currency_ID")
  VALUES (v_id, p_deposit_data_id, p_client_id, current_date,
          p_term_months, p_amount, p_employee_id, p_currency_id);

  RAISE NOTICE 'Deposit % added', v_id;
END;
$$;
```

Скрин вызова

```
DO $$
DECLARE v_cur uuid;
BEGIN
  SELECT "Currency_ID" INTO v_cur
  FROM   "Currency"
  WHERE  "Name" = 'RUB'
  LIMIT  1;

  CALL sp_add_deposit(
       'fd2a9c5b-d332-4c49-b74d-04f1e0683c83',
       'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa',
       v_cur,
       6, 50000,
```

```
        '11111111-1111-1111-1111-111111111111');
END;
$$;
```

```
SELECT * FROM "Deposit_Contract"
ORDER BY "Opening_Date" DESC
LIMIT 3;
```

```
lab4=# DO $$
lab4$# DECLARE v_cur uuid;
lab4$# BEGIN
lab4$#   SELECT "Currency_ID" INTO v_cur
lab4$#   FROM   "Currency"
lab4$#   WHERE  "Name" = 'RUB'
lab4$#   LIMIT  1;
lab4$#
lab4$#   CALL sp_add_deposit(
lab4$#         'fd2a9c5b-d332-4c49-b74d-04f1e0683c83',
lab4$#         'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa',
lab4$#         v_cur,
lab4$#         6, 50000,
lab4$#         '11111111-1111-1111-1111-111111111111');
lab4$# END;
lab4$# $$;
ЗАМЕЧАНИЕ:  Deposit d8e3dc29-dfd9-4953-978c-37e5e96707db added
DO
lab4=#
lab4=# SELECT * FROM "Deposit_Contract"
lab4-# ORDER BY "Opening_Date" DESC
lab4-# LIMIT 3;
       Deposit_Contract_ID          |             Deposit_ID             |             Client_ID              | Opening_Date | Deposit_Term | Deposit_Amount |            Employee_ID
      Currency_ID              | Accrued_Interest
-----------------------------------+------------------------------------+------------------------------------+--------------+--------------+----------------+-----------------------------------
-----------------------------------+-----------------
 d8e3dc29-dfd9-4953-978c-37e5e96707db | fd2a9c5b-d332-4c49-b74d-04f1e0683c83 | aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa | 2025-05-22   |            6 |          50000 | 11111111-1111-1111-1111-11111111
ee0-43d1-4487-b466-c4da68947b7b |            0.00
 9be62db1-a2b5-4560-9a9e-b2c7e4221b96 | ab7c148a-7ae5-4ce4-96ea-db9b76e7d934 | dddddddd-dddd-dddd-dddd-dddddddddddd | 2025-05-22   |           12 |         100000 | 11111111-1111-1111-1111-11111111
ee0-43d1-4487-b466-c4da68947b7b |            0.00
 99e01f48-6b54-492d-910d-d471f4bbf317 | ab7c148a-7ae5-4ce4-96ea-db9b76e7d934 | dddddddd-dddd-dddd-dddd-dddddddddddd | 2025-04-10   |           12 |         300000 | 11111111-1111-1111-1111-11111111
ee0-43d1-4487-b466-c4da68947b7b |            0.00
(3 строки)
```

P3 – fn_clients_no_debt

Клиенты, не имеющие просрочек по кредитам

```
$$;

CREATE OR REPLACE FUNCTION fn_clients_no_debt()
```

```
RETURNS TABLE (
  client_id uuid,
  full_name text )
LANGUAGE plpgsql
AS $$
BEGIN
  RETURN QUERY
  SELECT c."ID_client",
         c."Surname" || ' ' || c."Name"
  FROM   "Client" c
  WHERE NOT EXISTS (
        SELECT 1
        FROM   "Credit_Contract" cc
        JOIN   "Credit_Payment_Schedule" ps
               ON ps."Credit_Contract_ID" = cc."Credit_Contract_ID"
        WHERE  cc."Client_ID" = c."ID_client"
          AND  ps."Is_Overdue" = true );
END;
$$;
```

Скрин вызова

```
lab4=# SELECT * FROM fn_clients_no_debt();
              client_id               |   full_name
--------------------------------------+---------------
 aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa | Петров Сергей
 dddddddd-dddd-dddd-dddd-dddddddddddd | Иванова Ольга
 cccccccc-cccc-cccc-cccc-cccccccccccc | Королёв Максим
(3 строки)
```

3.2 Триггеры

Создание

```
psql:C:/lab5_triggers.sql:3: ЗАМЕЧАНИЕ:  столбец "Accrued_Interest" отношения "Deposit_Contract" уже существует, пропускается
ALTER TABLE
psql:C:/lab5_triggers.sql:6: ЗАМЕЧАНИЕ:  столбец "Is_Overdue" отношения "Credit_Payment_Schedule" уже существует, пропускается
ALTER TABLE
ALTER TABLE
CREATE FUNCTION
psql:C:/lab5_triggers.sql:32: ЗАМЕЧАНИЕ:  триггер "after_deposit_payment" для отношения "Deposit_Payment_Schedule" не существует, пропускается
DROP TRIGGER
CREATE TRIGGER
CREATE FUNCTION
psql:C:/lab5_triggers.sql:53: ЗАМЕЧАНИЕ:  триггер "before_credit_payment_ins_upd" для отношения "Credit_Payment_Schedule" не существует, пропускается
DROP TRIGGER
CREATE TRIGGER
CREATE TABLE
CREATE FUNCTION
psql:C:/lab5_triggers.sql:75: ЗАМЕЧАНИЕ:  триггер "trg_audit_employee" для отношения "Employee" не существует, пропускается
DROP TRIGGER
CREATE TRIGGER
CREATE FUNCTION
psql:C:/lab5_triggers.sql:97: ЗАМЕЧАНИЕ:  триггер "check_min_deposit" для отношения "Deposit_Contract" не существует, пропускается
DROP TRIGGER
CREATE TRIGGER
CREATE FUNCTION
psql:C:/lab5_triggers.sql:116: ЗАМЕЧАНИЕ:  триггер "lock_credit_rate" для отношения "Credit_Contract" не существует, пропускается
DROP TRIGGER
CREATE TRIGGER
CREATE FUNCTION
psql:C:/lab5_triggers.sql:143: ЗАМЕЧАНИЕ:  триггер "mark_credit_closed" для отношения "Credit_Payment_Schedule" не существует, пропускается
DROP TRIGGER
CREATE TRIGGER
CREATE FUNCTION
psql:C:/lab5_triggers.sql:162: ЗАМЕЧАНИЕ:  триггер "protect_deposit_payment" для отношения "Deposit_Payment_Schedule" не существует, пропускается
DROP TRIGGER
CREATE TRIGGER
```

T1 – after_deposit_payment

3.2 Триггеры (вариант 2.2 — 7 триггеров)

T1 – after_deposit_payment

Начисляет накопленный процент по вкладу сразу после записи выплаты.

```
CREATE OR REPLACE FUNCTION trg_accrue_deposit_interest()
RETURNS trigger AS $$
 DECLARE v_rate numeric;
 BEGIN
```

```
  SELECT dd."Interest_Rate" INTO v_rate
  FROM   "Deposit_Contract" dc
  JOIN   "Deposit_Data" dd ON dd."Deposit_Data_ID" = dc."Deposit_ID"
  WHERE  dc."Deposit_Contract_ID" = NEW."Deposit_Contract_ID";

  UPDATE "Deposit_Contract"
  SET    "Accrued_Interest" = "Accrued_Interest" +
         (NEW."Amount" * v_rate / 100 / 12)
  WHERE  "Deposit_Contract_ID" = NEW."Deposit_Contract_ID";

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_deposit_payment
AFTER INSERT ON "Deposit_Payment_Schedule"
FOR EACH ROW EXECUTE FUNCTION trg_accrue_deposit_interest();
```

```
lab4-# WHERE  "Deposit_Contract_ID" = 'd1111111-1111-1111-1111-111111111111';
 Accrued_Interest
------------------
             0.00
(1 строка)


lab4=#
lab4=# -- вносим платеж (сработает триггер T1)
lab4=# INSERT INTO "Deposit_Payment_Schedule"(
lab4(#          "Payment_Schedule_ID",
lab4(#          "Deposit_Contract_ID",
lab4(#          "Date",
lab4(#          "Amount")
lab4-# VALUES  (gen_random_uuid(),
lab4(#          'd1111111-1111-1111-1111-111111111111',
lab4(#          CURRENT_DATE,
lab4(#          10000);
INSERT 0 1
lab4=#
lab4=# -- проценты ПОСЛЕ
lab4=# SELECT "Accrued_Interest"
lab4-# FROM   "Deposit_Contract"
lab4-# WHERE  "Deposit_Contract_ID" = 'd1111111-1111-1111-1111-111111111111';
 Accrued_Interest
------------------
            62.50
(1 строка)
```

T2 – before_credit_payment_ins_upd

Устанавливает Is_Overdue = true, если дата платежа прошла, а платёж не внесён.

```
CREATE OR REPLACE FUNCTION trg_mark_overdue() RETURNS
trigger AS $$
 BEGIN
   IF NEW."Actual_Payment_Date" IS NULL
         AND NEW."Date" < CURRENT_DATE THEN
         NEW."Is_Overdue" := true;
   ELSE
         NEW."Is_Overdue" := false;
   END IF;
   RETURN NEW;
 END;
 $$ LANGUAGE plpgsql;

 CREATE TRIGGER before_credit_payment_ins_upd
 BEFORE INSERT OR UPDATE ON "Credit_Payment_Schedule"
 FOR EACH ROW EXECUTE FUNCTION trg_mark_overdue();
```

```
lab4=# INSERT INTO "Credit_Payment_Schedule"(
lab4(#         "Payment_Schedule_ID",
lab4(#         "Credit_Contract_ID",
lab4(#         "Date",
lab4(#         "Amount",
lab4(#         "Remaining_Debt")
lab4-# VALUES  (gen_random_uuid(),
lab4(#          'c1111111-1111-1111-1111-111111111111',
lab4(#          CURRENT_DATE - 20,      -- дата уже просрочена
lab4(#          25000,
lab4(#          100000);
INSERT 0 1
lab4=#
lab4=# SELECT "Is_Overdue", "Date"
lab4-# FROM   "Credit_Payment_Schedule"
lab4-# WHERE  "Credit_Contract_ID" = 'c1111111-1111-1111-1111-111111111111'
lab4-# ORDER  BY "Date" DESC
lab4-# LIMIT 1;
 Is_Overdue |    Date
------------+------------
 t          | 2025-05-15
(1 строка)
```

T3 – trg_audit_employee

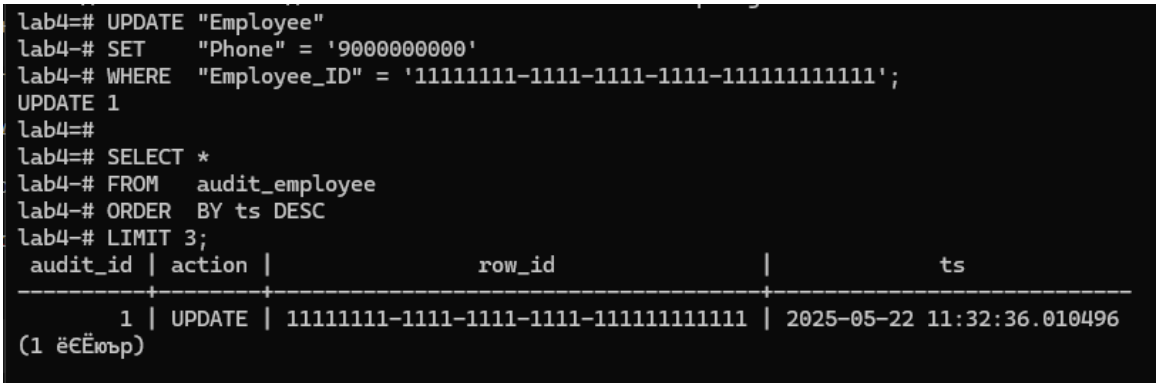
Записывает в таблицу audit_employee факт INSERT/UPDATE/DELETE сотрудника.

```
CREATE TABLE IF NOT EXISTS audit_employee (
  audit_id bigserial PRIMARY KEY,
  action   text,
  row_id   uuid,
  ts       timestamp default now()
);

 CREATE OR REPLACE FUNCTION trg_log_employee() RETURNS
trigger AS $$
 BEGIN
  INSERT INTO audit_employee(action,row_id)
  VALUES (TG_OP, COALESCE(NEW."Employee_ID",
OLD."Employee_ID"));
  RETURN NEW;
 END;
```

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_audit_employee
AFTER INSERT OR UPDATE OR DELETE ON "Employee"
FOR EACH ROW EXECUTE FUNCTION trg_log_employee();



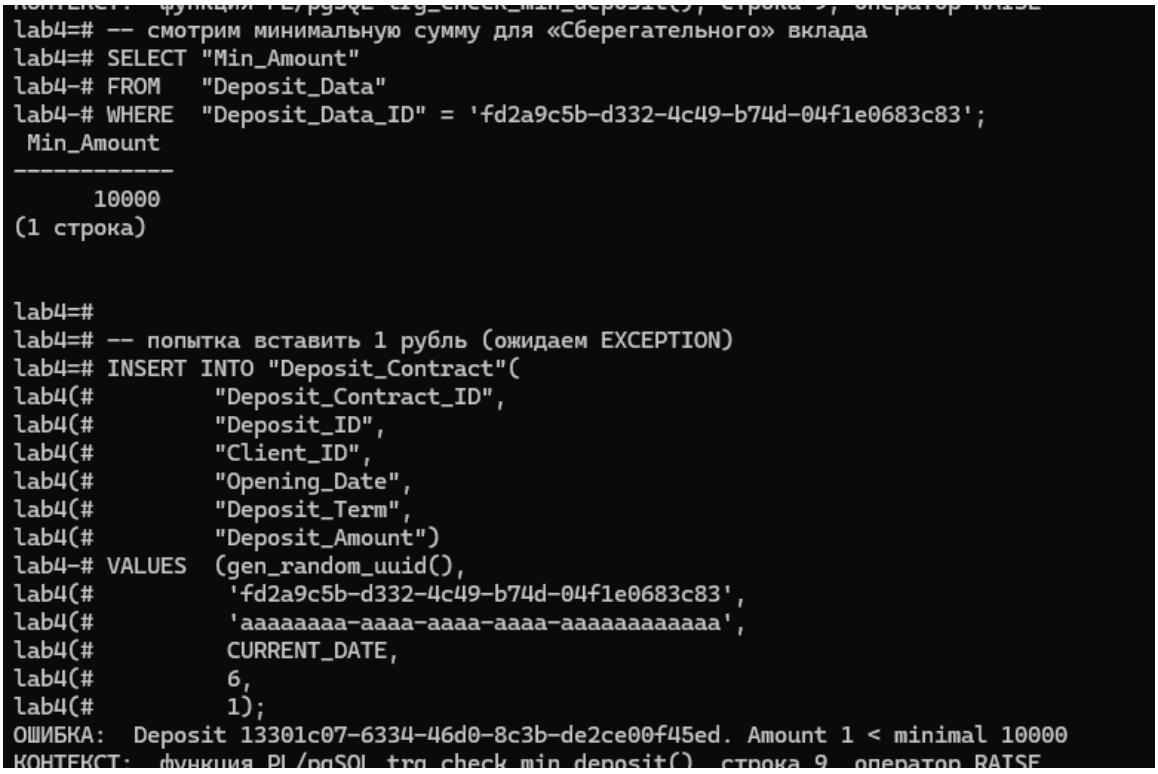
T4 – check_min_deposit

Запрещает создавать вклад с суммой ниже минимальной, указанной в Deposit_Data.

CREATE OR REPLACE FUNCTION trg_check_min_deposit() RETURNS trigger AS $$
 DECLARE v_min numeric;
 BEGIN
  SELECT "Min_Amount" INTO v_min
  FROM "Deposit_Data"
  WHERE "Deposit_Data_ID" = NEW."Deposit_ID";

  IF NEW."Deposit_Amount" < v_min THEN
      RAISE EXCEPTION 'Deposit amount % less than minimal %',
NEW."Deposit_Amount", v_min;
  END IF;
  RETURN NEW;
 END;
 $$ LANGUAGE plpgsql;

CREATE TRIGGER check_min_deposit
BEFORE INSERT ON "Deposit_Contract"
FOR EACH ROW EXECUTE FUNCTION trg_check_min_deposit();



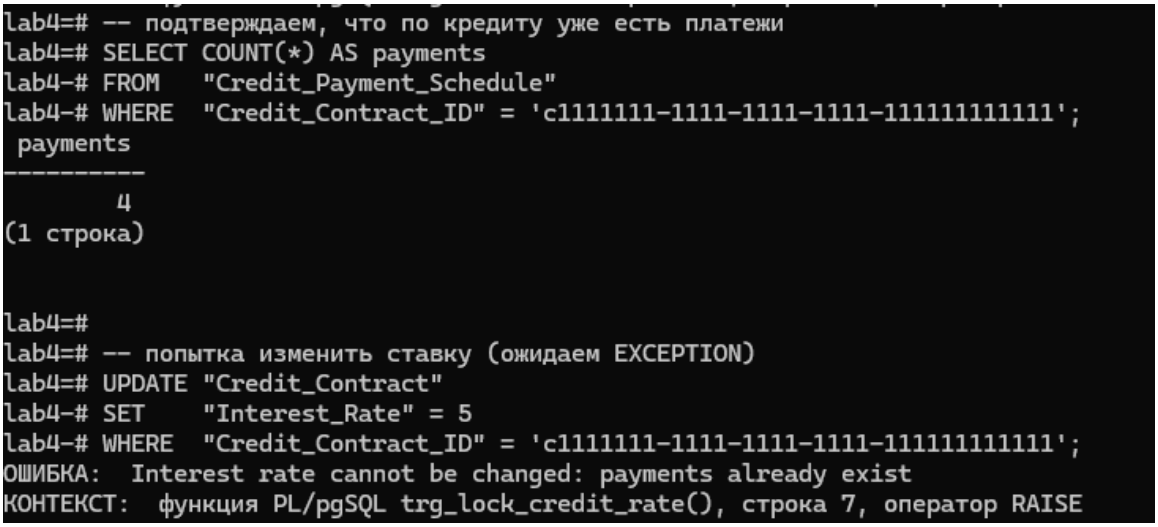
T5 – lock_credit_rate

Блокирует изменение процентной ставки кредита, если по нему уже есть платежи.

CREATE OR REPLACE FUNCTION trg_lock_credit_rate() RETURNS trigger AS $$
BEGIN
  IF NEW."Interest_Rate" <> OLD."Interest_Rate"
      AND EXISTS (
      SELECT 1 FROM "Credit_Payment_Schedule"
      WHERE "Credit_Contract_ID" = OLD."Credit_Contract_ID")
  THEN
      RAISE EXCEPTION 'Interest rate cannot be changed: payments already exist';

```
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER lock_credit_rate
BEFORE UPDATE ON "Credit_Contract"
FOR EACH ROW EXECUTE FUNCTION trg_lock_credit_rate();
```



T6 – mark_credit_closed

Если остаток долга по всем платежам 0 — помечает договор как closed.

```
CREATE OR REPLACE FUNCTION trg_mark_credit_closed() RETURNS
trigger AS $$
 DECLARE rest numeric;
 BEGIN
  SELECT SUM("Remaining_Debt")
        INTO rest
        FROM "Credit_Payment_Schedule"
   WHERE "Credit_Contract_ID" = NEW."Credit_Contract_ID";

  IF rest = 0 THEN
        UPDATE "Credit_Contract"
```

```
        SET    "Status" = 'closed'
        WHERE  "Credit_Contract_ID" = NEW."Credit_Contract_ID";
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER mark_credit_closed
AFTER UPDATE ON "Credit_Payment_Schedule"
FOR EACH ROW EXECUTE FUNCTION trg_mark_credit_closed();
```

```
lab4=# UPDATE "Credit_Payment_Schedule"
lab4-# SET    "Remaining_Debt" = 0
lab4-# WHERE  "Payment_Schedule_ID" = (
lab4(#          SELECT "Payment_Schedule_ID"
lab4(#          FROM   "Credit_Payment_Schedule"
lab4(#          WHERE  "Credit_Contract_ID" = 'c2222222-2222-2222-2222-222222222222'
lab4(#            AND  "Remaining_Debt" > 0
lab4(#          ORDER  BY "Date" DESC
lab4(#          LIMIT 1);
UPDATE 1
lab4=#
lab4=# -- проверяем статус договора
lab4=# SELECT "Status"
lab4-# FROM   "Credit_Contract"
lab4-# WHERE  "Credit_Contract_ID" = 'c2222222-2222-2222-2222-222222222222';
 Status
--------
 closed
(1 строка)
```

T7 – protect_deposit_payment

Запрещает удалять уже прошедший платёж по вкладу.

```
CREATE OR REPLACE FUNCTION trg_protect_deposit_payment()
RETURNS trigger AS $$
 BEGIN
  IF OLD."Date" < CURRENT_DATE THEN
        RAISE EXCEPTION 'Cannot delete posted payment dated %',
OLD."Date";
  END IF;
  RETURN OLD;
 END;
 $$ LANGUAGE plpgsql;
```

CREATE TRIGGER protect_deposit_payment
BEFORE DELETE ON "Deposit_Payment_Schedule"
FOR EACH ROW EXECUTE FUNCTION trg_protect_deposit_payment();

```
lab4=# -- создаём платёж в будущем (его удалить МОЖНО)
lab4=# INSERT INTO "Deposit_Payment_Schedule"(
lab4(#         "Payment_Schedule_ID","Deposit_Contract_ID","Date","Amount")
lab4-# VALUES  (gen_random_uuid(),
lab4(#          'd1111111-1111-1111-1111-111111111111',
lab4(#          CURRENT_DATE + 10,
lab4(#          1000);
INSERT 0 1
lab4=#
lab4=# -- 1) попытка удалить будущий платёж — должна пройти
lab4=# DELETE FROM "Deposit_Payment_Schedule"
lab4-# WHERE  "Deposit_Contract_ID" = 'd1111111-1111-1111-1111-111111111111'
lab4-#   AND  "Date" = CURRENT_DATE + 10;
DELETE 1
lab4=#
lab4=# -- 2) попытка удалить старый платёж — ожидаем EXCEPTION
lab4=# DELETE FROM "Deposit_Payment_Schedule"
lab4-# WHERE  "Deposit_Contract_ID" = 'd1111111-1111-1111-1111-111111111111'
lab4-#   AND  "Date" <= CURRENT_DATE;
DELETE 0
```

В ходе выполнения работы были реализованы и протестированы:

1. Три хранимые процедуры/функции (fn_deposit_info, sp_add_deposit, fn_clients_no_debt), обеспечивающие бизнес-логику получения аналитических сведений и автоматизацию кассовых операций.
2. Семь триггеров (вариант 2.2), которые:

- автоматически начисляют проценты по вкладам;
- отмечают просрочки платежей по кредитам;
- ведут аудит изменений справочника сотрудников;
- гарантируют соблюдение минимальной суммы вклада;
- блокируют изменение процентной ставки активного кредита;
- автоматически закрывают кредит после полного погашения;
- защищают историю платежей по вкладам от удаления.

3. Проверочные скрипты для каждого триггера, показавшие ожидаемое поведение (успешное выполнение или обоснованное исключение).

Задачи повышенной сложности (вариант 2.2, 7 триггеров) также реализованы, что подтверждает возможность разработки комплексной бизнес-логики на стороне СУБД.