Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по лабораторной работе «Процедуры, функции, триггеры в PostgreSQL»
по дисциплине «Проектирование и реализация баз данных»

Автор: Булыга Е.А.
Факультет: ИКТ
Группа: К32421
Преподаватель: Говорова М.М.

Санкт-Петербург 2023

Содержание

1 Введение

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

Практическое задание:

1. Создать процедуры/функции согласно индивидуальному заданию и (согласно индивидуальному заданию, часть 4).
2. Создать триггер для логирования событий вставки, удаления, редактирования данных в базе данных PostgreSQL (согласно индивидуальному заданию, часть 5). Допустимо создать универсальный триггер или отдельные триггеры на логирование действий.

2 Выполнение

2.1 Таблица capitalizations

Триггер на обновление баланса на счете при добавлении записи в таблицу капитализации вклада.

```
CREATE OR REPLACE FUNCTION fn_update_cash_after_capitalization()
RETURNS trigger LANGUAGE 'plpgsql'
AS $BODY$
    BEGIN
        UPDATE
          accounts
        SET
          balance = balance + NEW.money
        WHERE
          accounts.account_id =(
            SELECT
              account_id
            FROM
              deposits
            WHERE
              deposits.deposit_id = NEW.deposit_id
          );
        RETURN NEW;
    END;
$BODY$;

CREATE TRIGGER update_cash_after_capitalization
    AFTER INSERT
    ON capitalizations
    FOR EACH ROW
    EXECUTE FUNCTION fn_update_cash_after_capitalization();
```

Процедура добавления записей в таблицу капитализации.

```
CREATE OR REPLACE PROCEDURE proc_deposits_capitalization()
LANGUAGE 'plpgsql'
AS $BODY$
DECLARE
    i BIGINT;
BEGIN
    FOR i IN (
      SELECT
        deposit_id
      FROM
        deposits
    ) LOOP IF (
      (
        SELECT
          accounts.balance
        FROM
          (
            SELECT
```

```
          deposits.account_id,
          types_of_deposits.percent
        FROM
          deposits
          LEFT JOIN types_of_deposits
          ON deposits.type_of_deposit_id =
            types_of_deposits.type_of_deposit_id
        WHERE
          deposits.account_id = i
      ) AS tbl1
      LEFT JOIN accounts ON tbl1.account_id = accounts.account_id
  ) > 0
) THEN
INSERT INTO capitalizations (deposit_id, money)
VALUES
  (
    i,
    (
      SELECT
        (
          accounts.balance * tbl1.percent / 100 / 12
        )
      FROM
        (
          SELECT
            deposits.account_id,
            types_of_deposits.percent
          FROM
            deposits
            LEFT JOIN types_of_deposits
            ON deposits.type_of_deposit_id =
                types_of_deposits.type_of_deposit_id
          WHERE
            deposits.deposit_id = i
        ) AS tbl1
        LEFT JOIN accounts ON tbl1.account_id = accounts.account_id
    )
  );
ELSE
INSERT INTO capitalizations (deposit_id, money)
VALUES
  (
    i,
    (
      SELECT
        0.01
      FROM
        (
          SELECT
            deposits.account_id,
            types_of_deposits.percent
          FROM
            deposits
            LEFT JOIN types_of_deposits
            ON deposits.type_of_deposit_id =
                types_of_deposits.type_of_deposit_id
          WHERE
            deposits.deposit_id = i
        ) AS tbl1
        LEFT JOIN accounts ON tbl1.account_id = accounts.account_id
```

```
        )
      );
    END IF;
    END LOOP;
END;
$BODY$;
```

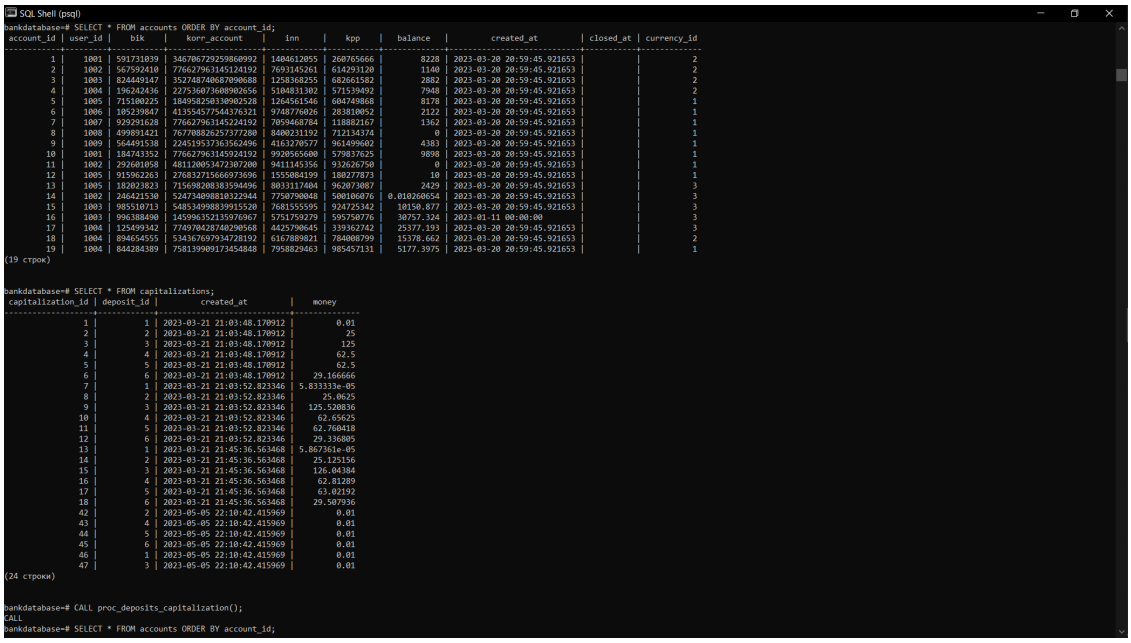

Рис. 1: До вызова процедуры

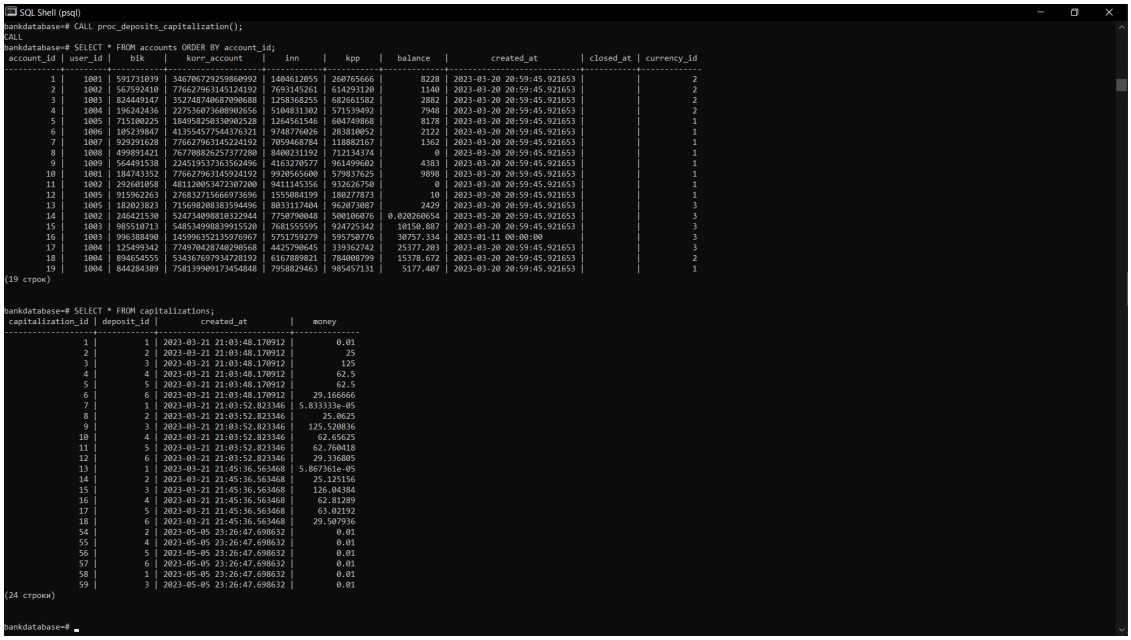

Рис. 2: После вызова процедуры

2.2 Таблица cash_turnover

Создадим триггер, который проверяет наличие необходимой суммы на балансе, если происходит снятие.

```
CREATE OR REPLACE FUNCTION
  fn_check_cash_before_turnover()
  RETURNS trigger
  LANGUAGE 'plpgsql'
AS $BODY$
    BEGIN
        IF (
          SELECT
            balance
          FROM
            accounts
          WHERE
            account_id = (
              SELECT
                account_id
              FROM
                cards
              WHERE
                card_number = NEW.card_number
            )
        ) < NEW.money AND NEW.in_out = '0' THEN
            RETURN NULL;
        END IF;
        RETURN NEW;
    END;
$BODY$;

CREATE TRIGGER check_cash_before_cash_turnover
    BEFORE INSERT
    ON cash_turnover
    FOR EACH ROW
    EXECUTE FUNCTION fn_check_cash_before_turnover();
```

Также добавим триггер, обновляющий баланс на счете в зависимости от действия: снятие или внесение.

```
CREATE OR REPLACE FUNCTION fn_update_cash_after_turnover()
    RETURNS trigger
    LANGUAGE 'plpgsql'
AS $BODY$
    BEGIN
        IF NEW.in_out = '1' THEN
            UPDATE
              accounts
            SET
              balance = balance + NEW.money
            WHERE
              account_id = (
                SELECT
                  account_id
                FROM
                  cards
                WHERE
                  card_number = NEW.card_number
              );
        ELSE
            UPDATE
              accounts
```

```
            SET
              balance = balance - NEW.money
            WHERE
              account_id = (
                SELECT
                  account_id
                FROM
                  cards
                WHERE
                  card_number = NEW.card_number
              );
        END IF;
        RETURN NEW;
    END;
$BODY$;

CREATE TRIGGER update_cash_after_turnover
    AFTER INSERT
    ON cash_turnover
    FOR EACH ROW
    EXECUTE FUNCTION fn_update_cash_after_turnover();
```

```
bankdatabase=# SELECT accounts.account_id, balance, card_number FROM cards LEFT JOIN accounts ON cards.account_id = accounts.account_id ORDER BY accounts.account_id;
 account_id | balance |   card_number
------------+---------+------------------
          1 |    8228 | 4858443003042444
          2 |    2140 | 4788927954158345
          3 |    2882 | 4712227257286898
          4 |    7948 | 4111194292302436
          5 |    8178 | 5582651815776708
          6 |    2122 | 6011008321403328
          7 |    1362 | 5222830188437785
          8 |       0 | 5577386779412118
          9 |    4383 | 4976693980888907
         10 |    9898 | 4325492597540086
         11 |       0 | 4404394795473009
         12 |      10 | 4308268238089647
         13 |    2429 | 5481196642315710
(13 строк)


bankdatabase=# INSERT INTO cash_turnover (card_number, in_out, nfc, address, money) VALUES (4788927954158345, '0', '0', '500-820 Fermentum Ave', 3000);
INSERT 0 0
```

Рис. 3: Неудачная попытка добавить запись о снятии наличных

```
bankdatabase=# SELECT accounts.account_id, balance, card_number FROM cards LEFT JOIN accounts ON cards.account_id = accounts.account_id ORDER BY accounts.account_id;
 account_id | balance |   card_number
------------+---------+------------------
          1 |    8228 | 4858443003042444
          2 |    2140 | 4788927954158345
          3 |    2882 | 4712227257286898
          4 |    7948 | 4111194292302436
          5 |    8178 | 5582651815776708
          6 |    2122 | 6011008321403328
          7 |    1362 | 5222830188437785
          8 |       0 | 5577386779412118
          9 |    4383 | 4976693980888907
         10 |    9898 | 4325492597540086
         11 |       0 | 4404394795473009
         12 |      10 | 4308268238089647
         13 |    2429 | 5481196642315710
(13 строк)


bankdatabase=# INSERT INTO cash_turnover (card_number, in_out, nfc, address, money) VALUES (4788927954158345, '0', '0', '500-820 Fermentum Ave', 3000);
INSERT 0 0
```

Рис. 4: Удачная попытка добавить запись о снятии наличных, а затем - внесении их обратно

2.3 Таблица payments

Создадим триггер, проверяющий наличие достаточной суммы для проведения операции оплаты в магазинах. Этот же триггер будет использоваться в операциях платы за обслуживание, т.к. в них используется номер карты.

```
CREATE OR REPLACE FUNCTION
  fn_check_cash_with_card_number()
  RETURNS trigger
  LANGUAGE 'plpgsql'
AS $BODY$
    BEGIN
        IF (
          SELECT
            balance
          FROM
            accounts
          WHERE
            account_id = (
              SELECT
                account_id
              FROM
                cards
              WHERE
                card_number = NEW.card_number
            )
        ) < NEW.money THEN
            RETURN NULL;
        END IF;
        RETURN NEW;
    END;
$BODY$;

CREATE TRIGGER check_cash_before_payment
    BEFORE INSERT
    ON cash_turnover
    FOR EACH ROW
    EXECUTE FUNCTION fn_check_cash_with_card_number();
```

Также создадим триггер, обновляющий информацию о балансе после внесения записи о трате в магазине.

```
CREATE OR REPLACE FUNCTION fn_update_cash_after_payment()
    RETURNS trigger
    LANGUAGE 'plpgsql'
AS $BODY$
    BEGIN
        UPDATE
          accounts
        SET
          balance = balance - NEW.money
        WHERE
          account_id = (
            SELECT
              account_id
            FROM
              cards
            WHERE
              card_number = NEW.card_number
          );
        RETURN NEW;
    END;
$BODY$;
```

```
CREATE TRIGGER update_cash_after_payment
    AFTER INSERT
    ON payments
    FOR EACH ROW
    EXECUTE FUNCTION fn_update_cash_after_payment();
```

```
bankdatabase=# SELECT accounts.account_id, balance, card_number FROM cards LEFT JOIN accounts ON cards.account_id = accounts.account_id ORDER BY accounts.account_id;
 account_id | balance |   card_number
------------+---------+-----------------
          1 |    8228 | 4858443003042444
          2 |    2140 | 4788927954158345
          3 |    2882 | 4712227257286898
          4 |    7948 | 4111194292302436
          5 |    8178 | 5582651815776708
          6 |    2122 | 6011008321403328
          7 |    1362 | 5222830188437785
          8 |       0 | 5577386779412118
          9 |    4383 | 4976693980888907
         10 |    9898 | 4325492597540086
         11 |       0 | 4404394795473009
         12 |      10 | 4308268238089647
         13 |    2429 | 5481196642315710
(13 строк)


bankdatabase=# INSERT INTO payments (retail_outlet_id, card_number, money) VALUES (4, 5481196642315710, 3000);
INSERT 0 0
bankdatabase=# INSERT INTO payments (retail_outlet_id, card_number, money) VALUES (4, 5481196642315710, 1500);
INSERT 0 1
bankdatabase=# SELECT accounts.account_id, balance, card_number FROM cards LEFT JOIN accounts ON cards.account_id = accounts.account_id ORDER BY accounts.account_id;
 account_id | balance |   card_number
------------+---------+-----------------
          1 |    8228 | 4858443003042444
          2 |    2140 | 4788927954158345
          3 |    2882 | 4712227257286898
          4 |    7948 | 4111194292302436
          5 |    8178 | 5582651815776708
          6 |    2122 | 6011008321403328
          7 |    1362 | 5222830188437785
          8 |       0 | 5577386779412118
          9 |    4383 | 4976693980888907
         10 |    9898 | 4325492597540086
         11 |       0 | 4404394795473009
         12 |      10 | 4308268238089647
         13 |     929 | 5481196642315710
(13 строк)
```

Рис. 5: Пример внесения записей в таблицу payments

2.4 Таблица payments_for_service

Добавим триггер на проверку достаточной суммы на балансе по функции, написанной в предыдущем пункте.

```
CREATE TRIGGER check_cash_before_service_fee
    BEFORE INSERT
    ON payments_for_service
    FOR EACH ROW
    EXECUTE FUNCTION fn_check_cash_with_card_number();
```

Создадим триггер для обновления информации о балансе при добавлении записи в таблицу о платах за обслуживание.

```
CREATE OR REPLACE FUNCTION fn_update_cash_after_service_fee()
    RETURNS trigger
    LANGUAGE 'plpgsql'
AS $BODY$
    BEGIN
        UPDATE
          accounts
        SET
          balance = balance - NEW.money
        WHERE
```

```
          account_id = (
            SELECT
              account_id
            FROM
              cards
            WHERE
              card_number = NEW.card_number
          );
        RETURN NEW;
    END;
$BODY$;

CREATE TRIGGER update_cash_after_service_fee
    AFTER INSERT
    ON payments_for_service
    FOR EACH ROW
    EXECUTE FUNCTION fn_update_cash_after_service_fee();
```

И также создадим процедуру, которая будет проходить по всем картам в базе, у которых плата за обслуживание отлична от нуля, и снимать платеж за обслуживание.

```
CREATE OR REPLACE PROCEDURE proc_payment_for_service()
    LANGUAGE 'plpgsql'
AS $BODY$
DECLARE
    i BIGINT;
BEGIN
    FOR i IN (
      SELECT
        card_number
      FROM
        cards
      WHERE
        type_of_card_id IN (
          SELECT
            type_of_card_id
          FROM
            types_of_cards
          WHERE
            service_fee <> 0
        )
    ) LOOP INSERT INTO payments_for_service (card_number, money)
    VALUES
      (
        i,
        (
          SELECT
            service_fee
          FROM
            cards
            LEFT JOIN types_of_cards
            ON cards.type_of_card_id = types_of_cards.type_of_card_id
          WHERE
            cards.card_number = i
        )
      );
    END LOOP;
```

```
END;
$BODY$;
```

```
bankdatabase=# SELECT account_id, balance, card_number, service_fee FROM (SELECT accounts.account_id, balance, card_number, type_of_card_id FROM cards LEFT JOIN accounts ON cards.account_id = accounts.account_id
) AS tbl1 LEFT JOIN types_of_cards ON tbl1.type_of_card_id = types_of_cards.type_of_card_id ORDER BY account_id;
 account_id | balance |    card_number     | service_fee
------------+---------+--------------------+-------------
          1 |    8078 | 4858443003042444 |         150
          2 |    2100 | 4788927954158345 |          40
          3 |    2882 | 4712227257286898 |           0
          4 |    7948 | 4111194292302436 |           0
          5 |    7178 | 5582651815776708 |        1000
          6 |    1272 | 6011008321403328 |         850
          7 |    1362 | 5222830188437785 |           0
          8 |       0 | 5577386779412118 |          60
          9 |    3533 | 4976693980888907 |         850
         10 |    9858 | 4325492597540086 |          40
         11 |       0 | 4404394795473009 |         150
         12 |      10 | 4308268238089647 |         200
         13 |     889 | 5481196642315710 |          40
(13 строк)


bankdatabase=# CALL proc_payment_for_service();
CALL
bankdatabase=# SELECT account_id, balance, card_number, service_fee FROM (SELECT accounts.account_id, balance, card_number, type_of_card_id FROM cards LEFT JOIN accounts ON cards.account_id = accounts.account_id
) AS tbl1 LEFT JOIN types_of_cards ON tbl1.type_of_card_id = types_of_cards.type_of_card_id ORDER BY account_id;
 account_id | balance |    card_number     | service_fee
------------+---------+--------------------+-------------
          1 |    7928 | 4858443003042444 |         150
          2 |    2060 | 4788927954158345 |          40
          3 |    2882 | 4712227257286898 |           0
          4 |    7948 | 4111194292302436 |           0
          5 |    6178 | 5582651815776708 |        1000
          6 |     422 | 6011008321403328 |         850
          7 |    1362 | 5222830188437785 |           0
          8 |       0 | 5577386779412118 |          60
          9 |    2683 | 4976693980888907 |         850
         10 |    9818 | 4325492597540086 |          40
         11 |       0 | 4404394795473009 |         150
         12 |      10 | 4308268238089647 |         200
         13 |     849 | 5481196642315710 |          40
(13 строк)
```

Рис. 6: До и после внесения записей в таблицу payments_for_service

2.5 Таблица remittances

Создадим триггер, проверяющий наличие необходимой суммы на счете отправителя.

```
CREATE OR REPLACE FUNCTION fn_check_cash_with_account_id()
    RETURNS trigger
    LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
    IF (
      SELECT
        balance
      FROM
        accounts
      WHERE
        account_id = NEW.out_account
    ) < NEW.money THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$BODY$;

CREATE TRIGGER check_cash_before_remittance
    BEFORE INSERT
    ON remittances
    FOR EACH ROW
    EXECUTE FUNCTION fn_check_cash_with_account_id();
```

И создадим триггер, вносящий изменения в баланс на счете как отправителя, так и получателя.

```
CREATE OR REPLACE FUNCTION
    fn_update_cash_after_remittance()
    RETURNS trigger LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
    UPDATE
      accounts
    SET
      balance = balance + NEW.money
    WHERE
      accounts.account_id = NEW.in_account;
    UPDATE
      accounts
    SET
      balance = balance - NEW.money
    WHERE
      accounts.account_id = NEW.out_account;
    RETURN NEW;
END;
$BODY$;

CREATE TRIGGER update_cash_after_remittance
    AFTER INSERT
    ON remittances
    FOR EACH ROW
    EXECUTE FUNCTION fn_update_cash_after_remittance();
```

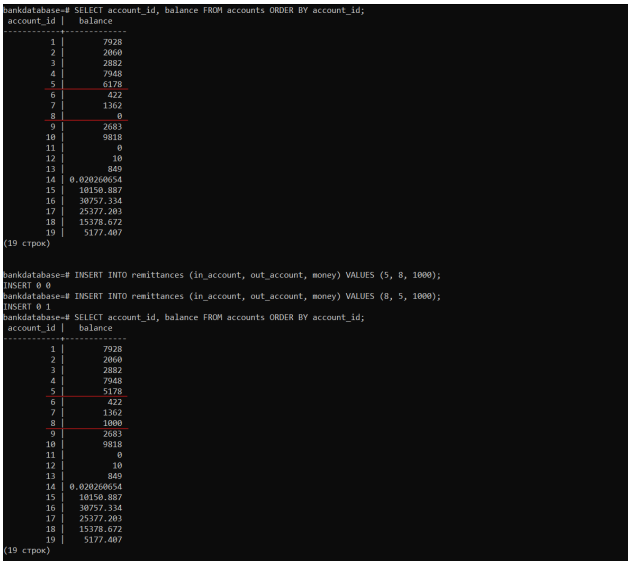

Рис. 7: Пример попыток добавления записей о переводах

Также мы бы хотели ограничить возможность переводов со вкладами согласно их типу. Создадим триггер, проверяющий возможность снятия или пополнения счета вклада.

```
CREATE OR REPLACE FUNCTION
    fn_check_access_to_update()
    RETURNS trigger
    LANGUAGE 'plpgsql'
AS $BODY$
    BEGIN
```

```
        IF NEW.out_account IN (
          SELECT
            account_id
          FROM
            deposits
        )
        AND (
          SELECT
            removable
          FROM
            deposits
            LEFT JOIN types_of_deposits
            ON deposits.type_of_deposit_id =
                types_of_deposits.type_of_deposit_id
          WHERE
            account_id = NEW.out_account
        ) = '1'
        OR NEW.in_account IN (
          SELECT
            account_id
          FROM
            deposits
        )
        AND (
          SELECT
            refillable
          FROM
            deposits
            LEFT JOIN types_of_deposits
            ON deposits.type_of_deposit_id =
                types_of_deposits.type_of_deposit_id
          WHERE
            account_id = NEW.in_account
        ) = '1' THEN RETURN NEW;
        END IF;
        RETURN NULL;
    END;
$BODY$;
```

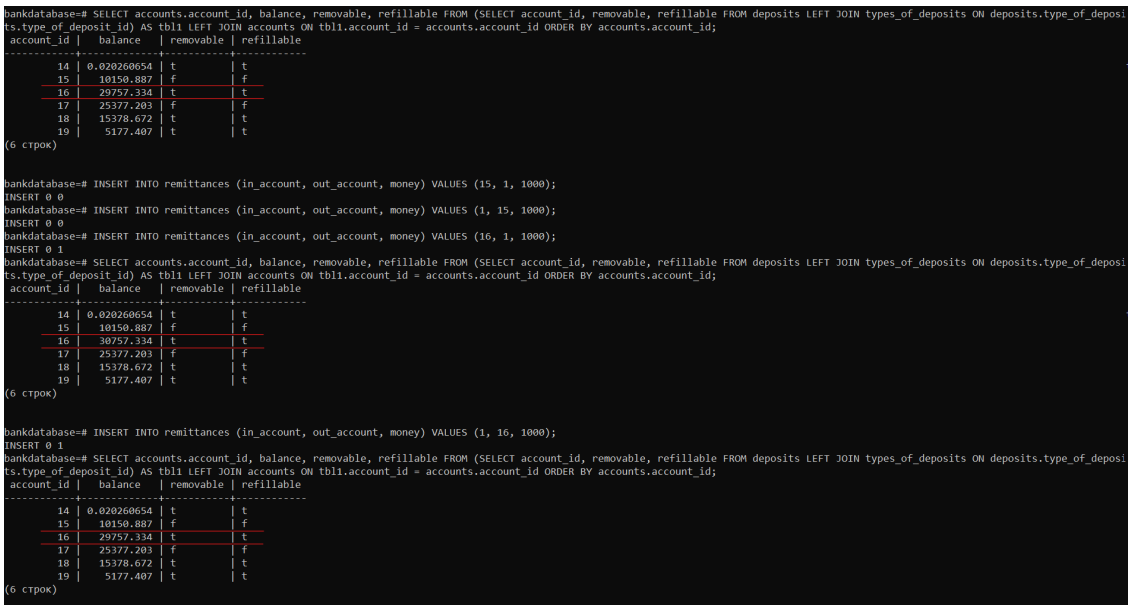

Рис. 8: Попытки совершить переводы с вкладами

3 Вывод

В процессе выполнения лабораторной работы были освоены практические навыки создания и использования функций, процедур и триггеров.