

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

**Отчет**

по лабораторной работе №3 «Процедуры, функции, триггеры в PostgreSQL»

по дисциплине «Проектирование и реализация баз данных»

Автор: Афолина Н.Р.

Факультет: ИКТ

Группа: К32421

Преподаватель: Говорова М.М.



Санкт-Петербург 2023

## Оглавление

Цель работы .....	3
Практическое задание .....	3
Вариант 19. БД «Банк».....	3
Вывод.....	10

## Цель работы

Овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

## Практическое задание

### Вариант 1

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.

### Вариант 19. БД «Банк»

#### Создать хранимые процедуры:

- о текущей сумме вклада и сумме начисленного за месяц процента для заданного клиента (в моем случае процента, который будет выплачен в конце срока вклада, так как у меня все вклады такого типа);

```
CREATE OR REPLACE FUNCTION get_balance_and_interest_saving(p_id_client
VARCHAR)
RETURNS TABLE(start_amount_saving_contract INT, interest_payment_saving INT)
AS $$
BEGIN
RETURN QUERY SELECT Sc.start_amount_saving_contract, Ssc.interest_payment_saving
FROM "Bank_schema".saving_contract AS Sc
JOIN "Bank_schema".schedule_saving_contract AS Ssc
ON Sc.id_saving_contract=Ssc.id_saving_contract
WHERE Sc.id_client = p_id_client;
END;
$$ LANGUAGE plpgsql;
```

```

Bank_Database=# CREATE OR REPLACE FUNCTION get_balance_and_interest_saving(p_id_client VARCHAR)
Bank_Database=# RETURNS TABLE(start_amount_saving_contract INT, interest_payment_saving INT)
Bank_Database=# AS $$
Bank_Database=# BEGIN
Bank_Database=# RETURN QUERY SELECT Sc.start_amount_saving_contract, Ssc.interest_payment_saving
Bank_Database=# FROM "Bank_schema".saving_contract AS Sc
Bank_Database=# JOIN "Bank_schema".schedule_saving_contract AS Ssc
Bank_Database=# ON Sc.id_saving_contract=Ssc.id_saving_contract
Bank_Database=# WHERE Sc.id_client = p_id_client;
Bank_Database=# END;
Bank_Database=# $$ LANGUAGE plpgsql;
CREATE FUNCTION
Bank_Database=# SELECT * FROM get_balance_and_interest_saving('12345678909');
-[ RECORD 1 ]-----+-----
start_amount_saving_contract | 30000
interest_payment_saving      | 6300
-[ RECORD 2 ]-----+-----
start_amount_saving_contract | 8000
interest_payment_saving      | 1600

Bank_Database=#

```

Рис. 1 – «Работа первой функции»

- найти клиента банка, имеющего максимальное количество кредитов на текущий день;

```
CREATE OR REPLACE FUNCTION get_client_with_max_loans()
```

```
RETURNS TABLE(name_client VARCHAR, number_of_loans INT) AS $$
```

```
BEGIN
```

```
RETURN QUERY
```

```
SELECT Cl.name_client, COUNT (Lc.id_client)::INT AS number_of_loans
```

```
FROM "Bank_schema".client Cl JOIN "Bank_schema".loan_contract Lc ON
Cl.id_client=Lc.id_client
```

```
WHERE Lc.startdate_loan_contract <= CURRENT_DATE AND Lc.enddate_loan_contract
>= CURRENT_DATE
```

```
GROUP BY Cl.name_client
```

```
HAVING COUNT (Lc.id_client) = (SELECT MAX(subquery.loan_count) FROM (SELECT
COUNT(id_client) AS loan_count FROM "Bank_schema".loan_contract GROUP BY
id_client) AS subquery);
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

```

Bank_Database=# CREATE OR REPLACE FUNCTION get_client_with_max_loans()
Bank_Database=# RETURNS TABLE(name_client VARCHAR, number_of_loans INT) AS $$
Bank_Database=# BEGIN
Bank_Database=# RETURN QUERY
Bank_Database=# SELECT Cl.name_client, COUNT (Lc.id_client)::INT AS number_of_loans
Bank_Database=# FROM "Bank_schema".client Cl JOIN "Bank_schema".loan_contract Lc ON Cl.id_client=Lc.id_client
Bank_Database=# WHERE Lc.startdate_loan_contract <= CURRENT_DATE AND Lc.enddate_loan_contract >= CURRENT_DATE
Bank_Database=# GROUP BY Cl.name_client
Bank_Database=# HAVING COUNT (Lc.id_client) = (SELECT MAX(subquery.loan_count) FROM (SELECT COUNT(id_client) AS loan_cou
nt FROM "Bank_schema".loan_contract GROUP BY id_client) AS subquery);
Bank_Database=# END;
Bank_Database=# $$ LANGUAGE plpgsql;
CREATE FUNCTION
Bank_Database=# SELECT * FROM get_client_with_max_loans();
-[ RECORD 1 ]-----
name_client | Hannah Marie Meloche
number_of_loans | 2
-[ RECORD 2 ]-----
name_client | John Smith
number_of_loans | 2
Bank_Database=#

```

Рис. 2 – «Работа второй функции»

- найти клиентов банка, не имеющих задолженности по кредитам.

```

CREATE OR REPLACE FUNCTION get_clients_with_no_debt()
RETURNS TABLE(name_client VARCHAR)
AS $$
BEGIN
RETURN QUERY
SELECT DISTINCT Cl.name_client
FROM ((("Bank_schema".client Cl JOIN "Bank_schema".loan_contract Lc ON
Cl.id_client=Lc.id_client) JOIN "Bank_schema".schedule_loan_contract Slc ON
Lc.id_loan_contract=Slc.id_loan_contract)
WHERE state_payment_loan = true;
END;
$$ LANGUAGE plpgsql;

```

```

Bank_Database=# CREATE OR REPLACE FUNCTION get_clients_with_no_debt()
Bank_Database=# RETURNS TABLE(name_client VARCHAR)
Bank_Database=# AS $$
Bank_Database=# BEGIN
Bank_Database=# RETURN QUERY
Bank_Database=# SELECT DISTINCT Cl.name_client
Bank_Database=# FROM ((("Bank_schema".client Cl JOIN "Bank_schema".loan_contract Lc ON Cl.id_client=Lc.id_client) JOIN "B
ank_schema".schedule_loan_contract Slc ON Lc.id_loan_contract=Slc.id_loan_contract)
Bank_Database=# WHERE state_payment_loan = true;
Bank_Database=# END;
Bank_Database=# $$ LANGUAGE plpgsql;
CREATE FUNCTION
Bank_Database=# SELECT * FROM get_clients_with_no_debt();
-[ RECORD 1 ]-----
name_client | Hannah Marie Meloche
-[ RECORD 2 ]-----
name_client | Jane Doe
-[ RECORD 3 ]-----
name_client | Sarah Johnson
-[ RECORD 4 ]-----
name_client | John Smith
-[ RECORD 5 ]-----
name_client | Michael Lee
Bank_Database=#

```

Рис. 3 – «Работа третьей функции»

### Создать необходимые триггеры.

- Триггер на логирование вставки данных в таблицу Client

```
CREATE OR REPLACE FUNCTION log_client_insert()
RETURNS TRIGGER AS $$
BEGIN
INSERT INTO "Bank_schema".client_log(action_type, client_id, name_client, email_client,
log_date)
VALUES ('INSERT', NEW.id_client, NEW.name_client, NEW.email_client,
CURRENT_TIMESTAMP);
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER client_insert_trigger
AFTER INSERT ON "Bank_schema".client
FOR EACH ROW
EXECUTE FUNCTION log_client_insert();
```

```
Bank_Database=# INSERT INTO "Bank_schema".client (id_client, id_information_client, name_client, email_client, address_c
lient, phonenumber_client) VALUES ('YA12349088', 'Repubblica Italiana, Toscana, 1999-05-10', 'Jenny Willows', 'jenny@gma
il.com', 'Florence, Italy', '+3912670027');
INSERT 0 1
Bank_Database=# INSERT INTO "Bank_schema".client (id_client, id_information_client, name_client, email_client, address_c
lient, phonenumber_client) VALUES ('YA12349089', 'Repubblica Italiana, Lazio, 2000-12-03', 'Francesca Marca', 'marca@gma
il.com', 'Lazio, Italy', '+3912671135');
INSERT 0 1
Bank_Database=# INSERT INTO "Bank_schema".client (id_client, id_information_client, name_client, email_client, address_c
lient, phonenumber_client) VALUES ('YA12349090', 'Repubblica Italiana, Veneto, 1979-06-15', 'Yana De Luca', 'jenny@gmail
.com', 'Venezia, Italy', '+3912670457');
INSERT 0 1
Bank_Database=# SELECT * FROM "Bank_schema".client_log;
-[ RECORD 1 ]+-----
log_id      | 1
action_type | INSERT
client_id   | YA12349088
name_client | Jenny Willows
email_client| jenny@gmail.com
log_date    | 2023-04-19 20:18:40.762386
-[ RECORD 2 ]+-----
log_id      | 2
action_type | INSERT
client_id   | YA12349089
name_client | Francesca Marca
email_client| marca@gmail.com
log_date    | 2023-04-19 20:18:40.805785
-[ RECORD 3 ]+-----
log_id      | 3
action_type | INSERT
client_id   | YA12349090
name_client | Yana De Luca
email_client| jenny@gmail.com
log_date    | 2023-04-19 20:18:40.809482
```

Рис. 4 – «Работа триггера на insert»

- Триггер на логирование обновления данных в таблице Client

```
CREATE OR REPLACE FUNCTION log_client_update()
RETURNS TRIGGER AS $$
BEGIN
```

```

INSERT INTO "Bank_schema".client_log(action_type, client_id, name_client, email_client,
log_date)
VALUES ('UPDATE', OLD.id_client, NEW.name_client, NEW.email_client,
CURRENT_TIMESTAMP);
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER client_update_trigger
AFTER UPDATE ON "Bank_schema".client
FOR EACH ROW
EXECUTE FUNCTION log_client_update();

```

```

Bank_Database=# CREATE TRIGGER client_update_trigger
Bank_Database=# AFTER UPDATE ON "Bank_schema".client
Bank_Database=# FOR EACH ROW
Bank_Database=# EXECUTE FUNCTION log_client_update();
CREATE TRIGGER
Bank_Database=# UPDATE "Bank_schema".client SET name_client = 'Jenna Willows', email_client = 'jenna@gmail.com' WHERE id_client = 'YA12349088';
UPDATE 1
Bank_Database=# SELECT * FROM "Bank_schema".client_log;
-[ RECORD 1 ]+-----
log_id      | 1
action_type | INSERT
client_id   | YA12349088
name_client | Jenny Willows
email_client| jenny@gmail.com
log_date    | 2023-04-19 20:18:40.762386
-[ RECORD 2 ]+-----
log_id      | 2
action_type | INSERT
client_id   | YA12349089
name_client | Francesca Marca
email_client| marca@gmail.com
log_date    | 2023-04-19 20:18:40.805785
-[ RECORD 3 ]+-----
log_id      | 3
action_type | INSERT
client_id   | YA12349090
name_client | Yana De Luca
email_client| jenny@gmail.com
log_date    | 2023-04-19 20:18:40.809482
-[ RECORD 4 ]+-----
log_id      | 4
action_type | UPDATE
client_id   | YA12349088
name_client | Jenna Willows
email_client| jenna@gmail.com
log_date    | 2023-04-19 20:25:47.663742
Bank_Database=#

```

Рис. 5 – «Работа триггера на update»

- Триггер на логирование удаления данных в таблице Client

```

CREATE OR REPLACE FUNCTION log_client_delete()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO "Bank_schema".client_log(action_type, client_id, name_client, email_client,
log_date)
VALUES ('DELETE', OLD.id_client, OLD.name_client, OLD.email_client,
CURRENT_TIMESTAMP);

```

```

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER client_delete_trigger
AFTER DELETE ON "Bank_schema".client
FOR EACH ROW
EXECUTE FUNCTION log_client_delete();

```

The screenshot shows a SQL Shell (psql) window with the following content:

```

Bank_Database=# CREATE TRIGGER client_delete_trigger
Bank_Database=# AFTER DELETE ON "Bank_schema".client
Bank_Database=# FOR EACH ROW
Bank_Database=# EXECUTE FUNCTION log_client_delete();
CREATE TRIGGER
Bank_Database=# DELETE FROM "Bank_schema".client WHERE id_client = 'YA12349089';
DELETE 1
Bank_Database=# SELECT * FROM "Bank_schema".client_log;
-[ RECORD 1 ]+-----
log_id      | 1
action_type | INSERT
client_id   | YA12349088
name_client | Jenny Willows
email_client| jenny@gmail.com
log_date    | 2023-04-19 20:18:40.762386
-[ RECORD 2 ]+-----
log_id      | 2
action_type | INSERT
client_id   | YA12349089
name_client | Francesca Marca
email_client| marca@gmail.com
log_date    | 2023-04-19 20:18:40.805785
-[ RECORD 3 ]+-----
log_id      | 3
action_type | INSERT
client_id   | YA12349090
name_client | Yana De Luca
email_client| jenny@gmail.com
log_date    | 2023-04-19 20:18:40.809482
-[ RECORD 4 ]+-----
log_id      | 4
action_type | UPDATE
client_id   | YA12349088
name_client | Jenna Willows
email_client| jenna@gmail.com
log_date    | 2023-04-19 20:25:47.663742
-[ RECORD 5 ]+-----
log_id      | 5
action_type | DELETE
client_id   | YA12349089
name_client | Francesca Marca
email_client| marca@gmail.com
log_date    | 2023-04-19 20:44:42.953843

```

Рис. 6 – «Работа триггера на delete»

- Триггер, который автоматически рассчитывает окончательную сумму для кредитного договора на основе суммы кредиты, процентной ставки и срока кредита.

```

CREATE OR REPLACE FUNCTION calculate_loan_payment()
RETURNS TRIGGER AS $$
DECLARE
    loan_amount INTEGER;
    interest_rate NUMERIC;
    loan_term VARCHAR;

```



```

        loan_term_int INTEGER;
        monthly_rate NUMERIC;
        monthly_payment NUMERIC;
BEGIN

        loan_amount := NEW.amount_loan_contract;
        SELECT loan.interest_loan INTO interest_rate FROM "Bank_schema".loan loan JOIN
        "Bank_schema".loan_contract contract ON loan.id_loan = contract.id_loan WHERE
        contract.id_loan_contract = NEW.id_loan_contract;
        SELECT loan.terms_loan INTO loan_term FROM "Bank_schema".loan loan JOIN
        "Bank_schema".loan_contract contract ON loan.id_loan = contract.id_loan WHERE
        contract.id_loan_contract = NEW.id_loan_contract;

        loan_term_int := substring(regex_replace(loan_term, '^[^d]', ', ', 'g'), '\d+'):INTEGER;
        monthly_rate := interest_rate / 12;
        monthly_payment := (loan_amount * monthly_rate) / (1 - POWER(1 + monthly_rate, -
        loan_term_int * 12));

        UPDATE "Bank_schema".loan_contract SET total_amount_loan_contract = loan_amount +
        monthly_payment * loan_term_int WHERE id_loan_contract = NEW.id_loan_contract;

        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER loan_payment_trigger
AFTER INSERT ON "Bank_schema".loan_contract
FOR EACH ROW
EXECUTE FUNCTION calculate_loan_payment();

```

```

CREATE FUNCTION
Bank_Database=#
Bank_Database=# INSERT INTO "Bank_schema".loan_contract(id_loan_contract, startdate_loan_contract, id_client, id_loan, id_worker, amount_loan_contract
, enddate_loan_contract, real_enddate_loan_contract, interest_loan_contract)
Bank_Database=# SELECT nextval('ls'), CURRENT_DATE, 'YA12349088', 6, 5, 70000, (CURRENT_DATE + INTERVAL '3 years'), NULL, interest_loan
Bank_Database=# FROM "Bank_schema".loan WHERE interest_loan IN (SELECT interest_loan FROM "Bank_schema".loan WHERE id_loan = 6);
INSERT 0 1
Bank_Database=# SELECT * FROM "Bank_schema".loan_contract WHERE id_client = 'YA12349088';
-[ RECORD 1 ]-----+-----
id_loan_contract      | 776
startdate_loan_contract | 2023-04-19
id_client              | YA12349088
id_loan                | 6
id_worker              | 5
amount_loan_contract   | 70000
total_amount_loan_contract | 280000
enddate_loan_contract  | 2026-04-19
real_enddate_loan_contract |
interest_loan_contract | 12
Bank_Database=#

```

Рис. 7 – «Работа личного триггера»

## Вывод

В данной лабораторной работе было выполнено создание и использование функций, так как их использование я посчитала более целесообразным для заданий моего варианта. Также я поработала с триггерами на логирование добавления, изменения и удаления данных и создала собственный триггер, который бы самостоятельно рассчитывал окончательную сумму кредита при добавлении нового контракта.