**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** Саид Наваф

**Факультет** прикладной информатики

**Группа** K3241

**Направление подготовки** 09.03.03 Прикладная информатика

**Образовательная программа** Мобильные и сетевые технологии 2023

**Преподаватель** Говорова Марина Михайловна

Санкт-Петербург

2025-2026

## 1 Цель работы

Овладеть практическими навыками 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

### 3.1 Вариант и название БД

**Вариант 12. БД «Прокат автомобилей»**

### 3.2 Процедуры

**№1**

**Задание:**

Создать процедуру для вывода данных о клиентах, которые арендовали автомобили в заданном временном интервале.

**Код процедуры:**

```
CREATE OR REPLACE FUNCTION get_clients_by_rental_period(
    start_date_param TIMESTAMP WITHOUT TIME ZONE,
    end_date_param TIMESTAMP WITHOUT TIME ZONE
)
```

```
RETURNS TABLE (
    client_id INTEGER,
    full_name VARCHAR(100),
    address VARCHAR(255),
    phone VARCHAR(20),
    passport VARCHAR(20),
    rental_start TIMESTAMP WITHOUT TIME ZONE,
    rental_end TIMESTAMP WITHOUT TIME ZONE
)
LANGUAGE SQL
AS $$
    SELECT
        c.id AS client_id,
        (c."фамилия" || ' ' || c."имя" || COALESCE(' ' || c."отчество", '')) AS full_name,
        c."адрес" AS address,
        c."номер_телефона" AS phone,
        c."номер_паспорта" AS passport,
        d."дата_начала_аренды" AS rental_start,
        d."дата_фактического_возврата" AS rental_end
    FROM
        "Клиент" c
    JOIN
        "Договор" d ON c.id = d."id_клиента"
    WHERE
        d."дата_начала_аренды" BETWEEN start_date_param AND end_date_param
    ORDER BY
        c.id, d."дата_начала_аренды";
$$;
```

SELECT * FROM get_clients_by_rental_period('2023-08-01 00:00:00', '2023-08-31 23:59:59');

## Скриншоты выполнения:

Query | Query History

```sql
CREATE OR REPLACE FUNCTION get_clients_by_rental_period(
    start_date_param TIMESTAMP WITHOUT TIME ZONE,
    end_date_param TIMESTAMP WITHOUT TIME ZONE
)
RETURNS TABLE (
    client_id INTEGER,
    full_name VARCHAR(100),
    address VARCHAR(255),
    phone VARCHAR(20),
    passport VARCHAR(20),
    rental_start TIMESTAMP WITHOUT TIME ZONE,
    rental_end TIMESTAMP WITHOUT TIME ZONE
)
LANGUAGE SQL
AS $$
    SELECT
        c.id AS client_id,
        (c."фамилия" || ' ' || c."имя" || COALESCE(' ' || c."отчество", '')) AS full_name,
        c."адрес" AS address,
        c."номер_телефона" AS phone,
        c."номер_паспорта" AS passport,
        d."дата_начала_аренды" AS rental_start,
        d."дата_фактического_возврата" AS rental_end
```

Query | Query History

```sql
        d."дата_фактического_возврата" AS rental_end
    FROM
        "Клиент" c
    JOIN
        "Договор" d ON c.id = d."id_клиента"
    WHERE
        d."дата_начала_аренды" BETWEEN start_date_param AND end_date_param
    ORDER BY
        c.id, d."дата_начала_аренды";
$$;

SELECT * FROM get_clients_by_rental_period('2023-08-01 00:00:00', '2023-08-31 23:59:59');
```

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1

| | client_id integer | full_name character varying | address character varying | phone character varying | passport character varying | rental_start timestamp without time zone | rental_end timestamp without time zone |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Иванов Иван Иванович | ул. Ленина, 10, кв.5 | +79111234567 | 1234567890 | 2023-08-01 12:00:00 | 2023-08-10 17:30:00 |
| 2 | 2 | Наваф Саид | ул. Примерная, 123 | +79112233445 | 1122334455 | 2023-08-05 13:00:00 | 2023-08-15 19:30:00 |
| 3 | 3 | Сидоров Алексей | ул. Гагарина, 5, кв.33 | +79031234567 | 4567890123 | 2023-08-10 11:00:00 | [null] |

## №2

## Задание:

Создать процедуру, выводящую сведения о том, какие автомобили арендовал клиент по заданному номеру телефона.

## Код процедуры:

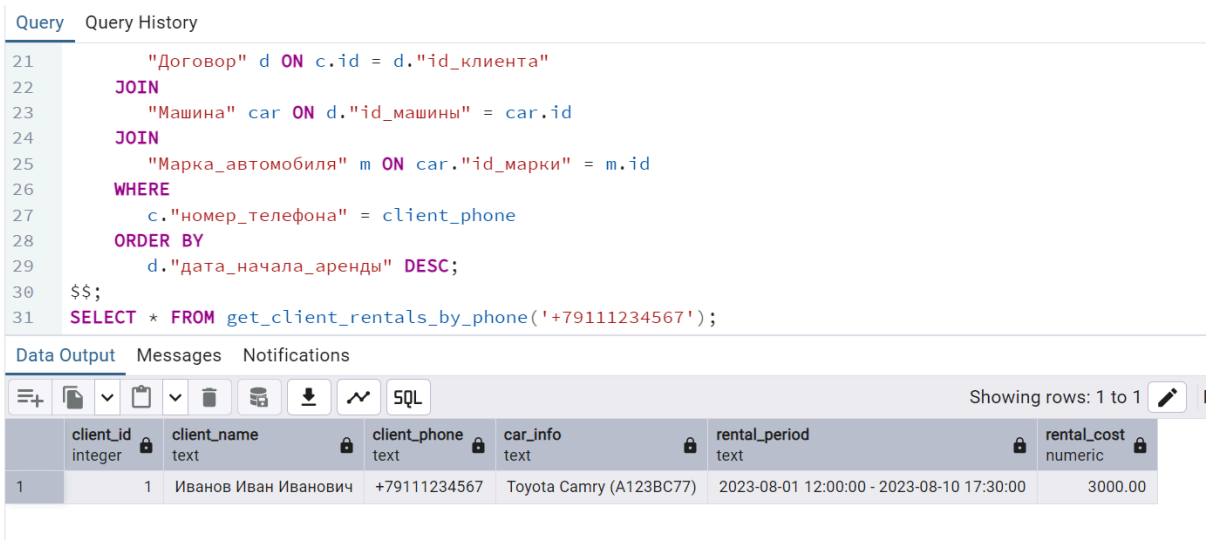
```
CREATE OR REPLACE FUNCTION get_client_rentals_by_phone(client_phone VARCHAR(20))
RETURNS TABLE (
    client_id INTEGER,
    client_name TEXT,
    client_phone TEXT,
    car_info TEXT,
    rental_period TEXT,
    rental_cost NUMERIC
)
LANGUAGE SQL AS $$
    SELECT
        c.id,
        c."фамилия" || ' ' || c."имя" || COALESCE(' ' || c."отчество", ''),
        c."номер_телефона",
        m."название_модели" || ' (' || car."регистрационный_номер" || ')',
        d."дата_начала_аренды" || ' - ' || COALESCE(d."дата_фактического_возврата"::TEXT, 'еще не возвращена'),
        d."цена_аренды"
    FROM
            "Клиент" c
    JOIN
            "Договор" d ON c.id = d."id_клиента"
    JOIN
            "Машина" car ON d."id_машины" = car.id
    JOIN
            "Марка_автомобиля" m ON car."id_марки" = m.id
    WHERE
            c."номер_телефона" = client_phone
    ORDER BY
            d."дата_начала_аренды" DESC;
```

$$;

SELECT * FROM get_client_rentals_by_phone('+79111234567');

## Скриншоты выполнения:

```
Query  Query History
1  CREATE OR REPLACE FUNCTION get_client_rentals_by_phone(client_phone VARCHAR(20))
2  RETURNS TABLE (
3      client_id INTEGER,
4      client_name TEXT,
5      client_phone TEXT,
6      car_info TEXT,
7      rental_period TEXT,
8      rental_cost NUMERIC
9  )
10 LANGUAGE SQL AS $$
11     SELECT
12         c.id,
13         c."фамилия" || ' ' || c."имя" || COALESCE(' ' || c."отчество", ''),
14         c."номер_телефона",
15         m."название_модели" || ' (' || car."регистрационный_номер" || ')',
16         d."дата_начала_аренды" || ' - ' || COALESCE(d."дата_фактического_возврата"::TEXT, 'еще не возвращена'),
17         d."цена_аренды"
18     FROM
19        "Клиент" c
20     JOIN
21        "Договор" d ON c.id = d."id_клиента"
22     JOIN
```



```
Query  Query History
21         "Договор" d ON c.id = d."id_клиента"
22      JOIN
23         "Машина" car ON d."id_машины" = car.id
24      JOIN
25         "Марка_автомобиля" m ON car."id_марки" = m.id
26      WHERE
27         c."номер_телефона" = client_phone
28      ORDER BY
29         d."дата_начала_аренды" DESC;
30 $$;
31 SELECT * FROM get_client_rentals_by_phone('+79111234567');
```

Data Output  Messages  Notifications

Showing rows: 1 to 1

| | client_id integer | client_name text | client_phone text | car_info text | rental_period text | rental_cost numeric |
|---|---|---|---|---|---|---|
| 1 | 1 | Иванов Иван Иванович | +79111234567 | Toyota Camry (A123BC77) | 2023-08-01 12:00:00 - 2023-08-10 17:30:00 | 3000.00 |

## №3

## Задание:

Создать процедуру для вычисления суммарного дохода компании за указанный месяц.

## Код процедуры:

CREATE OR REPLACE FUNCTION calculate_monthly_revenue(

```
    month_year VARCHAR(7)
)
RETURNS NUMERIC(12,2)
LANGUAGE plpgsql
AS $$
DECLARE
    total_revenue NUMERIC(12,2);
BEGIN
    SELECT SUM(d."цена_аренды")
    INTO total_revenue
    FROM "Договор" d
    WHERE
        TO_CHAR(d."дата_начала_аренды", 'YYYY-MM') = month_year
        OR (
            TO_CHAR(d."дата_начала_аренды", 'YYYY-MM') <= month_year
            AND (
                d."дата_фактического_возврата" IS NULL
                OR TO_CHAR(d."дата_фактического_возврата", 'YYYY-MM') >= month_year
            )
        );

    RETURN COALESCE(total_revenue, 0.00);
END;
$$;
SELECT calculate_monthly_revenue('2023-08');
```

## Скриншоты выполнения:

Query  Query History

```sql
CREATE OR REPLACE FUNCTION calculate_monthly_revenue(
    month_year VARCHAR(7)
)
RETURNS NUMERIC(12,2)
LANGUAGE plpgsql
AS $$
DECLARE
    total_revenue NUMERIC(12,2);
BEGIN
    SELECT SUM(d."цена_аренды")
    INTO total_revenue
    FROM "Договор" d
    WHERE
        TO_CHAR(d."дата_начала_аренды", 'YYYY-MM') = month_year
        OR (
            TO_CHAR(d."дата_начала_аренды", 'YYYY-MM') <= month_year
            AND (
                d."дата_фактического_возврата" IS NULL
                OR TO_CHAR(d."дата_фактического_возврата", 'YYYY-MM') >= month_year
            )
        );
```

```sql
        );

    RETURN COALESCE(total_revenue, 0.00);
END;
$$;
SELECT calculate_monthly_revenue('2023-08');
```

Data Output  Messages  Notifications

| | calculate_monthly_revenue numeric |
|---|---|
| 1 | 10500.00 |

## 3.3 Триггеры

**№1**

### Описание триггера:

Создать триггер для логирования всех изменений в таблице "Договор" в отдельную таблицу audit_log

### Код триггера:

```
CREATE TABLE договор_аудит (
```

```
    id SERIAL PRIMARY KEY,
    operation_type VARCHAR(10) NOT NULL,
    column_name VARCHAR(50),
    договор_id INTEGER NOT NULL,
    old_value TEXT,
    new_value TEXT,
    change_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    user_name TEXT DEFAULT CURRENT_USER
);

CREATE OR REPLACE FUNCTION log_договор_changes()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF TG_OP = 'UPDATE' THEN
        IF OLD."id_клиента" IS DISTINCT FROM NEW."id_клиента" THEN
            INSERT INTO договор_аудит (operation_type, column_name, договор_id, old_value, new_value)
            VALUES ('UPDATE', 'id_клиента', OLD.id, OLD."id_клиента"::TEXT, NEW."id_клиента"::TEXT);
        END IF;

        IF OLD."id_машины" IS DISTINCT FROM NEW."id_машины" THEN
            INSERT INTO договор_аудит (operation_type, column_name, договор_id, old_value, new_value)
            VALUES ('UPDATE', 'id_машины', OLD.id, OLD."id_машины"::TEXT, NEW."id_машины"::TEXT);
        END IF;

        IF OLD."цена_аренды" IS DISTINCT FROM NEW."цена_аренды" THEN
            INSERT INTO договор_аудит (operation_type, column_name, договор_id, old_value, new_value)
            VALUES ('UPDATE', 'цена_аренды', OLD.id, OLD."цена_аренды"::TEXT, NEW."цена_аренды"::TEXT);
```

```
        END IF;

    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO договор_аудит (operation_type, договор_id, old_value)
        VALUES ('DELETE', OLD.id,
                json_build_object(
                    'id_клиента', OLD."id_клиента",
                    'id_машины', OLD."id_машины",
                    'дата_начала_аренды', OLD."дата_начала_аренды",
                    'цена_аренды', OLD."цена_аренды"
                )::TEXT);

    ELSIF TG_OP = 'INSERT' THEN
        INSERT INTO договор_аудит (operation_type, договор_id, new_value)
        VALUES ('INSERT', NEW.id,
                json_build_object(
                    'id_клиента', NEW."id_клиента",
                    'id_машины', NEW."id_машины",
                    'дата_начала_аренды', NEW."дата_начала_аренды",
                    'цена_аренды', NEW."цена_аренды"
                )::TEXT);
    END IF;

    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER договор_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON "Договор"
```

```
FOR EACH ROW EXECUTE FUNCTION log_договор_changes();

INSERT INTO "Договор" (
    "id_клиента",
    "id_машины",
    "дата_заключения_договора",
    "дата_начала_аренды",
    "цена_аренды",
    "id_сотрудника"
) VALUES (
    1,
    1,
    CURRENT_TIMESTAMP,
    '2025-01-01 10:00:00',
    5000.00,
    1
);


UPDATE "Договор"
SET "цена_аренды" = 5500.00
WHERE id = (SELECT MAX(id) FROM "Договор");


DELETE FROM "Договор"
WHERE id = (SELECT MAX(id) FROM "Договор");


SELECT * FROM договор_аудит ORDER BY change_date DESC;
```

## Скриншоты выполнения:

Query | Query History

```sql
CREATE TABLE договор_аудит (
    id SERIAL PRIMARY KEY,
    operation_type VARCHAR(10) NOT NULL,
    column_name VARCHAR(50),
    договор_id INTEGER NOT NULL,
    old_value TEXT,
    new_value TEXT,
    change_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    user_name TEXT DEFAULT CURRENT_USER
);

CREATE OR REPLACE FUNCTION log_договор_changes()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF TG_OP = 'UPDATE' THEN
        IF OLD."id_клиента" IS DISTINCT FROM NEW."id_клиента" THEN
            INSERT INTO договор_аудит (operation_type, column_name, договор_id, old_value, new_value)
            VALUES ('UPDATE', 'id_клиента', OLD.id, OLD."id_клиента"::TEXT, NEW."id_клиента"::TEXT);
        END IF;

        IF OLD."id_машины" IS DISTINCT FROM NEW."id_машины" THEN
```

Total rows: Query complete 00:00:00.095

Query | Query History

```sql
        IF OLD."id_машины" IS DISTINCT FROM NEW."id_машины" THEN
            INSERT INTO договор_аудит (operation_type, column_name, договор_id, old_value, new_value)
            VALUES ('UPDATE', 'id_машины', OLD.id, OLD."id_машины"::TEXT, NEW."id_машины"::TEXT);
        END IF;

        IF OLD."цена_аренды" IS DISTINCT FROM NEW."цена_аренды" THEN
            INSERT INTO договор_аудит (operation_type, column_name, договор_id, old_value, new_value)
            VALUES ('UPDATE', 'цена_аренды', OLD.id, OLD."цена_аренды"::TEXT, NEW."цена_аренды"::TEXT);
        END IF;

    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO договор_аудит (operation_type, договор_id, old_value)
        VALUES ('DELETE', OLD.id,
               json_build_object(
                   'id_клиента', OLD."id_клиента",
                   'id_машины', OLD."id_машины",
                   'дата_начала_аренды', OLD."дата_начала_аренды",
                   'цена_аренды', OLD."цена_аренды"
               )::TEXT);

    ELSIF TG_OP = 'INSERT' THEN
        INSERT INTO договор_аудит (operation_type, договор_id, new_value)
        VALUES ('INSERT', NEW.id,
```

Total rows: Query complete 00:00:00.095

```
43      ELSIF TG_OP = 'INSERT' THEN
44          INSERT INTO договор_аудит (operation_type, договор_id, new_value)
45          VALUES ('INSERT', NEW.id,
46                  json_build_object(
47                      'id_клиента', NEW."id_клиента",
48                      'id_машины', NEW."id_машины",
49                      'дата_начала_аренды', NEW."дата_начала_аренды",
50                      'цена_аренды', NEW."цена_аренды"
51                  )::TEXT);
52      END IF;
53
54      RETURN NEW;
55  END;
56  $$;
57
58  CREATE OR REPLACE TRIGGER договор_audit_trigger
59  AFTER INSERT OR UPDATE OR DELETE ON "Договор"
60  FOR EACH ROW EXECUTE FUNCTION log_договор_changes();
```

Data Output | Messages | Notifications

```
CREATE TRIGGER

Query returned successfully in 95 msec.
```

Total rows: | Query complete 00:00:00.095

Query | Query History

```sql
INSERT INTO "Договор" (
    "id_клиента",
    "id_машины",
    "дата_заключения_договора",
    "дата_начала_аренды",
    "цена_аренды",
    "id_сотрудника"
) VALUES (
    1,
    1,
    CURRENT_TIMESTAMP,
    '2025-01-01 10:00:00',
    5000.00,
    1
);

UPDATE "Договор"
SET "цена_аренды" = 5500.00
WHERE id = (SELECT MAX(id) FROM "Договор");

DELETE FROM "Договор"
WHERE id = (SELECT MAX(id) FROM "Договор");
```

Data Output | Messages | Notifications

Total rows: 3 | Query complete 00:00:00.200

```sql
UPDATE "Договор"
SET "цена_аренды" = 5500.00
WHERE id = (SELECT MAX(id) FROM "Договор");

DELETE FROM "Договор"
WHERE id = (SELECT MAX(id) FROM "Договор");

SELECT * FROM договор_аудит ORDER BY change_date DESC;
```

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1 of 1

| | id [PK] integer | operation_type character varying (10) | column_name character varying (50) | договор_id integer | old_value text | new_value text |
|---|---|---|---|---|---|---|
| 1 | 1 | INSERT | [null] | 9 | [null] | {"id_клиента" |
| 2 | 2 | UPDATE | цена_аренды | 9 | 5000.00 | 5500.00 |
| 3 | 3 | DELETE | [null] | 9 | {"id_клиента" : 1, "id_машины" : 1, "дата_начала_аренды" : "2025-01-01T10:00:00", "цена_аренды" : 5500.00} | [null] |

**№2**

## Описание триггера:

Триггер для автоматического обновления статуса автомобиля при аварии (перевод в ремонт и расчет времени простоя).

## Код триггера:

```
CREATE OR REPLACE FUNCTION update_car_status_after_accident()
RETURNS TRIGGER AS $$
DECLARE
    repair_days INTEGER;
BEGIN
    IF NEW.статус_аварии = 'Авария' AND OLD.статус_аварии != 'Авария' THEN
        repair_days := FLOOR(NEW.коэффициент_уровня * 7);

        UPDATE "Машина"
        SET
            "дефектов" = "дефектов" + 1,
            "дата_последнего_техобслуживания" = CURRENT_DATE + repair_days
        WHERE id = (SELECT "id_машины" FROM "Договор" WHERE id = NEW."id_договора");

        INSERT INTO "Логи_ремонтов" (
            "id_аварии",
            "id_машины",
            "дата_начала_ремонта",
            "предполагаемая_дата_окончания"
        ) VALUES (
            NEW.id,
            (SELECT "id_машины" FROM "Договор" WHERE id = NEW."id_договора"),
            CURRENT_DATE,
```

```
            CURRENT_DATE + repair_days
        );
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update_car_status
AFTER UPDATE OF "статус_аварии" ON "Авария"
FOR EACH ROW
EXECUTE FUNCTION update_car_status_after_accident();

CREATE TABLE IF NOT EXISTS "Логи_ремонтов" (
    id SERIAL PRIMARY KEY,
    "id_аварии" INTEGER REFERENCES "Авария"(id),
    "id_машины" INTEGER REFERENCES "Машина"(id),
    "дата_начала_ремонта" DATE NOT NULL,
    "предполагаемая_дата_окончания" DATE NOT NULL,
    "фактическая_дата_окончания" DATE
);

UPDATE "Авария"
SET "статус_аварии" = 'Авария',
    "коэффициент_уровня" = 1.5
WHERE id = 1;

SELECT * FROM "Машина" WHERE id = (SELECT "id_машины" FROM "Договор" WHERE id = 2);
```

SELECT * FROM "Логи_ремонтов";

Query | Query History

```
CREATE OR REPLACE FUNCTION update_car_status_after_accident()
RETURNS TRIGGER AS $$
DECLARE
    repair_days INTEGER;
BEGIN
    IF NEW.статус_аварии = 'Авария' AND OLD.статус_аварии != 'Авария' THEN
        repair_days := FLOOR(NEW.коэффициент_уровня * 7);

        UPDATE "Машина"
        SET
            "дефектов" = "дефектов" + 1,
            "дата_последнего_техобслуживания" = CURRENT_DATE + repair_days
        WHERE id = (SELECT "id_машины" FROM "Договор" WHERE id = NEW."id_договора");

        INSERT INTO "Логи_ремонтов" (
            "id_аварии",
            "id_машины",
            "дата_начала_ремонта",
            "предполагаемая_дата_окончания"
        ) VALUES (
            NEW.id,
            (SELECT "id_машины" FROM "Договор" WHERE id = NEW."id_договора"),
            CURRENT_DATE,
```

Total rows: | Query complete 00:00:00.136

Query | Query History

```
            CURRENT_DATE,
            CURRENT_DATE + repair_days
        );
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update_car_status
AFTER UPDATE OF "статус_аварии" ON "Авария"
FOR EACH ROW
EXECUTE FUNCTION update_car_status_after_accident();
```

Data Output | Messages | Notifications

```
CREATE TRIGGER

Query returned successfully in 136 msec.
```

Query | Query History

```
CREATE TABLE IF NOT EXISTS "Логи_ремонтов" (
    id SERIAL PRIMARY KEY,
    "id_аварии" INTEGER REFERENCES "Авария"(id),
    "id_машины" INTEGER REFERENCES "Машина"(id),
    "дата_начала_ремонта" DATE NOT NULL,
    "предполагаемая_дата_окончания" DATE NOT NULL,
    "фактическая_дата_окончания" DATE
);

UPDATE "Авария"
SET "статус_аварии" = 'Авария',
    "коэффициент_уровня" = 1.5
WHERE id = 1;

SELECT * FROM "Машина" WHERE id = (SELECT "id_машины" FROM "Договор" WHERE id = 2);

SELECT * FROM "Логи_ремонтов";
```

Data Output | Messages | Notifications

Showing rows: 1 to 1

| | id [PK] integer | id_аварии integer | id_машины integer | дата_начала_ремонта date | предполагаемая_дата_окончания date | фактическая_дата_окончания date |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2 | 2025-05-20 | 2025-05-30 | [null] |

Total rows: 1 | Query complete 00:00:00.187

**№3**

## Описание триггера:

Триггер для контроля пересечения периодов аренды автомобилей. Обеспечивает, чтобы один автомобиль не мог быть арендован разными клиентами в один и тот же период времени. При попытке создать или изменить договор с пересекающимися датами система генерирует ошибку с подробным описанием конфликта.

## Код триггера:

```
CREATE OR REPLACE FUNCTION check_car_rental_period()

RETURNS TRIGGER AS $$

DECLARE

  conflict_info TEXT;

BEGIN


  IF NEW."дата_начала_аренды" IS NOT NULL AND NEW."дата_возврата" IS NOT NULL THEN


    SELECT INTO conflict_info

      'Договор №' || id || ' с ' || "дата_начала_аренды" || ' по ' || "дата_возврата"

    FROM "Договор"

    WHERE "id_машины" = NEW."id_машины"

    AND id != COALESCE(NEW.id, -1)

    AND (

      (NEW."дата_начала_аренды", NEW."дата_возврата") OVERLAPS

      ("дата_начала_аренды", "дата_возврата")

    )

    LIMIT 1;



    IF conflict_info IS NOT NULL THEN

      RAISE EXCEPTION
```

```
            'Автомобиль % уже арендован. %',
            NEW."id_машины",
            conflict_info;
        END IF;
    END IF;


    NEW."дата_заключения_договора" = COALESCE(NEW."дата_заключения_договора", NOW());
    NEW."есть_нарушения" = COALESCE(NEW."есть_нарушения", false);

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;



CREATE TRIGGER trg_check_car_rental_period
BEFORE INSERT OR UPDATE ON "Договор"
FOR EACH ROW
EXECUTE FUNCTION check_car_rental_period();
INSERT INTO "Договор" (
    "id_клиента", "id_машины", "дата_начала_аренды", "дата_возврата",
    "id_сотрудника", "цена_аренды"
) VALUES (
    2, 2, '2025-06-05', '2025-06-15',
    1, 6000.00
);
INSERT INTO "Договор" (
    "id_клиента", "id_машины", "дата_начала_аренды", "дата_возврата",
    "id_сотрудника", "цена_аренды"
```

) VALUES (

2, 1, '2025-06-21', '2025-06-25',

1, 6000.00

);

## Скриншоты выполнения:

Query Query History

```
1  CREATE OR REPLACE FUNCTION check_car_rental_period()
2  RETURNS TRIGGER AS $$
3  DECLARE
4      conflict_info TEXT;
5  BEGIN
6
7      IF NEW."дата_начала_аренды" IS NOT NULL AND NEW."дата_возврата" IS NOT NULL THEN
8
9          SELECT INTO conflict_info
10             'Договор №' || id || ' с ' || "дата_начала_аренды" || ' по ' || "дата_возврата"
11         FROM "Договор"
12         WHERE "id_машины" = NEW."id_машины"
13         AND id != COALESCE(NEW.id, -1)
14         AND (
15             (NEW."дата_начала_аренды", NEW."дата_возврата") OVERLAPS
16             ("дата_начала_аренды", "дата_возврата")
17         )
18         LIMIT 1;
19
20
21         IF conflict_info IS NOT NULL THEN
22             RAISE EXCEPTION
```

Data Output Messages Notifications

Total rows: Query complete 00:00:00.131

Query | Query History

```
    END IF;


    NEW."дата_заключения_договора" = COALESCE(NEW."дата_заключения_договора", NOW());
    NEW."есть_нарушения" = COALESCE(NEW."есть_нарушения", false);

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trg_check_car_rental_period
BEFORE INSERT OR UPDATE ON "Договор"
FOR EACH ROW
EXECUTE FUNCTION check_car_rental_period();
```

Data Output | Messages | Notifications

```
CREATE TRIGGER

Query returned successfully in 131 msec.
```

Query | Query History

```
INSERT INTO "Договор" (
    "id_клиента", "id_машины", "дата_начала_аренды", "дата_возврата",
    "id_сотрудника", "цена_аренды"
) VALUES (
    2, 2, '2025-06-05', '2025-06-15',
    1, 6000.00
);
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 56 msec.
```

Query Query History

```
INSERT INTO "Договор" (
    "id_клиента", "id_машины", "дата_начала_аренды", "дата_возврата",
    "id_сотрудника", "цена_аренды"
) VALUES (
    2, 1, '2025-06-21', '2025-06-25',
    1, 6000.00
);
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 57 msec.
```

## 4 Выводы

В рамках данной работы я получил практические навыки использования процедур, функций и триггеров в базе данных PostgreSQL, создав согласно заданию три процедуры и три триггера для своей базы данных.