

**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 (Синюков Лев Владимирович)

Факультет прикладной информатики

Группа __3240__

Направление подготовки 09.03.03 Прикладная информатика

Образовательная программа Мобильные и сетевые технологии 2023

Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2024/2025

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

Практическое задание (min - 6 баллов, max - 10 баллов, доп. баллы - 3):

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).

Дополнительные баллы - 3:

Модифицировать триггер (триггерную функцию) на проверку корректности входа и выхода сотрудника (см. Практическое задание 1 Лабораторного практикума (Приложение)) с максимальным учетом «узких» мест некорректных данных по входу и выходу).

Указание. Работа выполняется в консоли SQL Shell (psql).

Индивидуальное задание:

Вариант 10. БД «Автовокзал»

Описание предметной области: С автовокзала ежедневно отправляется несколько междугородных/международных автобусных рейсов. Номер рейса определяется маршрутом и временем отправления. По всем промежуточным остановкам на маршруте известны название, тип населенного пункта, время прибытия, отправления, время стоянки.

Автобусы курсируют по расписанию, но могут назначаться дополнительные рейсы на заданный период или определенные даты.

Билеты могут продаваться предварительно, но не ранее чем за 10 суток. Билет может быть приобретен в кассе автовокзала или онлайн. В билете указывается номер места в автобусе (необязательно). На каждый рейс может продаваться не более 10 билетов без места, цена на которые снижается на 10%. Пунктами отправления и назначения, согласно билету, могут быть промежуточные остановки.

Билеты могут продаваться в кассе автовокзала или онлайн.

Необходимо учитывать, что местом посадки и высадки пассажира могут быть промежуточные остановки согласно купленному билету.

На каждый рейс формируется экипаж из двух водителей. Необходимо хранить данные о прохождении медосмотра перед рейсом (дата, статус, причина недопуска).

БД должна содержать следующий минимальный набор сведений: Номер рейса. Номер водителя. Номер автобуса. Паспортные данные водителя. Пункт отправления. Пункт назначения. Промежуточные остановки. Дата отправления. Время отправления. Время в пути. Тип автобуса. Количество мест в автобусе. Страна. Производитель. Год выпуска. Номер билета. Номер места в автобусе (при наличии). Цена билета. ФИО пассажира. Паспортные данные пассажира.

Дополните состав атрибутов на основе анализа предметной области.

Задание 4. Создать хранимые процедуры:

- Продажи билета.
- Возврата билета.
- Добавления нового рейса.

Выполнение

Схема инфологической модели данных БД в нотации IDEF1X



Процедуры

1. Продажа билета

Создадим процедуру, в которой будет 3 IN параметра и 1 OUT. Также сделаем проверку, что место в автобусе не занято.

```
CREATE OR REPLACE PROCEDURE sell_ticket(  
    IN in_trip_id INT,  
    IN in_passport_passenger_id INT,  
    IN in_seat_number INT,  
    OUT out_ticket_id INT  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    IF EXISTS (  
        SELECT 1  
        FROM ticket  
        WHERE trip_id = in_trip_id  
            AND seat_number = in_seat_number  
            AND status = 'Purchased'  
    ) THEN
```

```

        RAISE EXCEPTION 'Место % занято для рейса %', in_seat_number,
in_trip_id;
    END IF;

    INSERT INTO ticket (trip_id, passport_passenger_id, seat_number,
status)
        VALUES (in_trip_id, in_passport_passenger_id, in_seat_number,
'Purchased')
    RETURNING ticket_id INTO out_ticket_id;
END;
$$;

```

Вызов процедуры:

```

DO $$
DECLARE t_id INT;
BEGIN
    CALL sell_ticket(8, 1, 7, t_id);
    RAISE NOTICE 'Новый билет ID = %', t_id;
END;
$$;

```

Результат работы:

```

CREATE PROCEDURE
bus_system=# DO $$
bus_system## DECLARE t_id INT;
bus_system## BEGIN
bus_system##     CALL sell_ticket(8, 1, 7, t_id);
bus_system##     RAISE NOTICE 'Новый билет ID = %', t_id;
bus_system## END;
bus_system## $$;
ЗАМЕЧАНИЕ: Новый билет ID = 26
DO
bus_system=# DO $$
bus_system## DECLARE t_id INT;
bus_system## BEGIN
bus_system##     CALL sell_ticket(8, 1, 7, t_id);
bus_system##     RAISE NOTICE 'Новый билет ID = %', t_id;
bus_system## END;
bus_system## $$;
ОШИБКА: Место 7 занято для рейса 8
КОНТЕКСТ: функция PL/pgSQL sell_ticket(integer,integer,integer), строка 10, оператор RAISE
SQL-оператор: "CALL sell_ticket(8, 1, 7, t_id)"
функция PL/pgSQL inline_code_block, строка 4, оператор CALL
bus_system=# |

```

2. Возврат билета

Создадим процедуру, в которой будет в качестве IN параметра ID билета, а в качестве OUT - статус возврата, возможно вернуть билет или нет.

```

CREATE OR REPLACE PROCEDURE return_ticket(
    IN in_ticket_id INT,
    OUT is_success BOOLEAN
)
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE ticket
    SET status = 'Cancelled'
    WHERE ticket_id = in_ticket_id
        AND status != 'Cancelled'
    RETURNING ticket_id INTO in_ticket_id;
    is_success := FOUND;
END;
$$;

```

Вызов процедуры:

```

DO $$
DECLARE
    result BOOLEAN;
BEGIN
    CALL return_ticket(26, result);
    RAISE NOTICE 'Возврат выполнен: %', result;
END;
$$;

```

```

bus_system=# $$,
CREATE PROCEDURE
bus_system=# DO $$
bus_system=# DECLARE
bus_system=#     result BOOLEAN;
bus_system=# BEGIN
bus_system=#     CALL return_ticket(26, result);
bus_system=#     RAISE NOTICE 'Возврат выполнен: %', result;
bus_system=# END;
bus_system=# $$;
ЗАМЕЧАНИЕ: Возврат выполнен: t

```

3. Добавления нового рейса

```

CREATE OR REPLACE PROCEDURE add_trip(
    IN in_route_id INT,
    IN in_bus_id INT,
    IN in_departure_date DATE,
    IN in_status VARCHAR,
    IN in_price NUMERIC,

```

```

        OUT out_trip_id INT
    )
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO trip (route_id, bus_id, departure_date, status, price)
        VALUES (in_route_id, in_bus_id, in_departure_date, in_status,
in_price)
    RETURNING trip_id INTO out_trip_id;
END;
$$;

```

Вызов процедуры:

```

DO $$
DECLARE
    t_id INT;
BEGIN
    CALL add_trip(1, 1, CURRENT_DATE + 5, 'Boarding', 100.00, t_id);
    RAISE NOTICE 'Добавлен новый рейс с ID = %', t_id;
END;
$$;

```

```

CREATE PROCEDURE
bus_system=# DO $$
bus_system$# DECLARE
bus_system$#     t_id INT;
bus_system$# BEGIN
bus_system$#     CALL add_trip(1, 1, CURRENT_DATE + 5, 'Boarding', 100.00, t_id);
bus_system$#     RAISE NOTICE 'Добавлен новый рейс с ID = %', t_id;
bus_system$# END;
bus_system$# $$;
ЗАМЕЧАНИЕ:  Добавлен новый рейс с ID = 17
DO

```

```

bus_system=# SELECT * FROM trip;
 trip_id | route_id | bus_id | departure_date | status | price
-----+-----+-----+-----+-----+-----
      1 |      1 |      1 | 2025-01-01 | Boarding | 1500.00
      2 |      1 |      1 | 2025-05-15 | Boarding | 1500.00
      3 |      2 |      1 | 2025-05-14 | Completed | 1800.00
      4 |      2 |      1 | 2025-05-10 | Completed | 1800.00
      5 |      3 |      2 | 2025-05-12 | Completed | 1000.00
      6 |      1 |      1 | 2025-05-14 | Boarding | 1500.00
      7 |      1 |      1 | 2025-05-18 | Boarding | 1000.00
      8 |      1 |      1 | 2025-12-15 | Completed | 100.00
      9 |      1 |      1 | 2026-01-10 | Completed | 1000.00
     10 |      1 |      1 | 2026-02-01 | Completed | 1000.00
     11 |      2 |      1 | 2026-01-15 | Completed | 2000.00
     12 |      2 |      1 | 2026-02-10 | Completed | 200.00
     13 |      2 |      1 | 2025-05-14 | Completed | 1300.00
     14 |      2 |      1 | 2025-05-14 | Completed | 1300.00
     15 |      2 |      1 | 2025-05-14 | Completed | 1300.00
     16 |      2 |      1 | 2025-05-14 | Completed | 1300.00
     17 |      1 |      1 | 2025-05-30 | Boarding | 100.00
(17 строк)

```

Триггеры

1. Автоматическая установка статуса Booked при выпуске билета

```

CREATE OR REPLACE FUNCTION set_ticket_status_booked()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.status IS NULL THEN
        NEW.status := 'Booked';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_ticket_default_status
BEFORE INSERT ON ticket
FOR EACH ROW
EXECUTE FUNCTION set_ticket_status_booked();

```

Проверка:

```

bus_system=# INSERT INTO ticket (trip_id, passport_passenger_id, seat_number) VALUES (1, 1, 72);
INSERT 0 1
bus_system=# SELECT ticket_id, status FROM ticket WHERE seat_number = 72;
 ticket_id | status
-----+-----
        29 | Booked
(1 строка)

```

2. Запрет на продажу билетов после отправления автобуса в рейс

```
CREATE OR REPLACE FUNCTION prevent_late_ticket_sale()
RETURNS TRIGGER AS $$
DECLARE
    trip_date DATE;
BEGIN
    SELECT departure_date INTO trip_date FROM trip WHERE trip_id =
NEW.trip_id;
    IF trip_date < CURRENT_DATE THEN
        RAISE EXCEPTION 'Автобус отправился в рейс, приобретение билетов
невозможно';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_ticket_sale_check
BEFORE INSERT ON ticket
FOR EACH ROW
EXECUTE FUNCTION prevent_late_ticket_sale();
```

Проверка:

```
bus_system=# INSERT INTO trip (route_id, bus_id, departure_date, status, price) VALUES (1, 1, CURRENT_DATE - 3, 'Completed', 1000.00);
INSERT 0 1
bus_system=# INSERT INTO ticket (trip_id, passport_passenger_id, seat_number) VALUES (18, 1, 88);
ОШИБКА: Автобус отправился в рейс, приобретение билетов невозможно
КОНТЕКСТ: функция PL/pgSQL prevent_late_ticket_sale(), строка 7, оператор RAISE
```

3. Установка статуса Completed на рейс в прошлом

```
CREATE OR REPLACE FUNCTION auto_complete_trip()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.departure_date < CURRENT_DATE THEN
        NEW.status := 'Completed';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_trip_auto_status
BEFORE INSERT ON trip
FOR EACH ROW
EXECUTE FUNCTION auto_complete_trip();
```

Проверка:


```

bus_system=# INSERT INTO trip (route_id, bus_id, departure_date, status, price) VALUES (1, 1, CURRENT_DATE - 2, NULL, 1000.00);
INSERT 0 1
bus_system=# SELECT * FROM trip;
 trip_id | route_id | bus_id | departure_date | status   | price
-----+-----+-----+-----+-----+-----
      1 |         |      1 | 2025-01-01     | Boarding | 1500.00
      2 |         |      1 | 2025-05-15     | Boarding | 1500.00
      3 |         |      2 | 2025-05-14     | Completed | 1800.00
      4 |         |      2 | 2025-05-10     | Completed | 1800.00
      5 |         |      3 | 2025-05-12     | Completed | 1000.00
      6 |         |      1 | 2025-05-14     | Boarding | 1500.00
      7 |         |      1 | 2025-05-18     | Boarding | 1000.00
      8 |         |      1 | 2025-12-15     | Completed | 100.00
      9 |         |      1 | 2026-01-10     | Completed | 1000.00
     10 |         |      1 | 2026-02-01     | Completed | 1000.00
     11 |         |      2 | 2026-01-15     | Completed | 2000.00
     12 |         |      2 | 2026-02-10     | Completed | 200.00
     13 |         |      2 | 2025-05-14     | Completed | 1300.00
     14 |         |      2 | 2025-05-14     | Completed | 1300.00
     15 |         |      2 | 2025-05-14     | Completed | 1300.00
     16 |         |      2 | 2025-05-14     | Completed | 1300.00
     17 |         |      1 | 2025-05-30     | Boarding | 100.00
     18 |         |      1 | 2025-05-25     | Completed | 1000.00
     19 |         |      1 | 2025-05-24     | Completed | 1200.00
     20 |         |      1 | 2025-05-23     | Completed | 1000.00
     23 |         |      1 | 2025-05-24     | Completed | 1000.00
(21 строка)

bus_system=# SELECT trip_id, status FROM trip ORDER BY trip_id DESC LIMIT 1;
 trip_id | status
-----+-----
      23 | Completed
(1 строка)

```

4. Логирование новых пассажиров

```

CREATE TABLE passenger_log (
    log_id SERIAL PRIMARY KEY,
    full_name TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE OR REPLACE FUNCTION log_new_passenger()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO passenger_log (full_name)
    VALUES (NEW.full_name);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_log_passenger
AFTER INSERT ON passenger
FOR EACH ROW
EXECUTE FUNCTION log_new_passenger();

```

Проверка:

```

bus_system=# INSERT INTO passenger (full_name) VALUES ('passenger');
INSERT 0 1
bus_system=# SELECT * FROM passenger_log ORDER BY log_id DESC LIMIT 1;
 log_id | full_name |          created_at
-----+-----+-----
      1 | passenger | 2025-05-26 01:28:03.157156
(1 строка)

```

5. Проверка, был ли пройден водителем медосмотр

```

CREATE OR REPLACE FUNCTION validate_crew_medical()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.medical_checkup_date != CURRENT_DATE THEN
        RAISE EXCEPTION 'Медосмотр не был пройден';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_crew_medical_check
BEFORE INSERT ON crew
FOR EACH ROW
EXECUTE FUNCTION validate_crew_medical();

```

Проверка:

```

bus_system=# INSERT INTO crew (trip_id, passport_driver_id, medical_status, medical_checkup_date, reason_id) VALUES (1, 1, 'Passed', CURRENT_DATE - 1, NULL);
ОШИБКА: Медосмотр не был пройден
КОНТЕКСТ: функция PL/pgSQL validate_crew_medical(), строка 4, оператор RAISE

```

6. Автоудаление билета при возврате

```

CREATE OR REPLACE FUNCTION delete_cancelled_ticket()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.status = 'Cancelled' THEN
        DELETE FROM ticket WHERE ticket_id = NEW.ticket_id;
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_delete_cancelled
AFTER UPDATE ON ticket
FOR EACH ROW
EXECUTE FUNCTION delete_cancelled_ticket();

```

Проверка:

```
bus_system=# SELECT * FROM ticket WHERE trip_id = 8 AND seat_number = 100;
 ticket_id | trip_id | passport_passenger_id | seat_number | status 
-----+-----+-----+-----+-----
          34 |         8 |             1 |         100 | Purchased
(1 строка)

bus_system=# UPDATE ticket SET status = 'Cancelled' WHERE trip_id = 8 AND seat_number = 100;
UPDATE 1
bus_system=# SELECT * FROM ticket WHERE trip_id = 8 AND seat_number = 100;
 ticket_id | trip_id | passport_passenger_id | seat_number | status 
-----+-----+-----+-----+-----
(0 строк)
```

7. Проверка данных ФИО пассажира на дублирование в таблице

```
CREATE OR REPLACE FUNCTION prevent_duplicate_passenger()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM passenger WHERE full_name = NEW.full_name
    ) THEN
        RAISE EXCEPTION 'Пассажир с такими ФИО уже существует';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_unique_passenger_name
BEFORE INSERT ON passenger
FOR EACH ROW
EXECUTE FUNCTION prevent_duplicate_passenger();
```

Проверка:

```
bus_system=# SELECT * FROM passenger;
 passenger_id | full_name 
-----+-----
            1 | Andrey Sahur
            2 | passenger
(2 строки)

bus_system=# INSERT INTO passenger (full_name) VALUES ('passenger');
ОШИБКА: Пассажир с такими ФИО уже существует
КОНТЕКСТ: функция PL/pgSQL prevent_duplicate_passenger(), строка 6, оператор RAISE
bus_system=# INSERT INTO passenger (full_name) VALUES ('Andrey Sahur');
ОШИБКА: Пассажир с такими ФИО уже существует
КОНТЕКСТ: функция PL/pgSQL prevent_duplicate_passenger(), строка 6, оператор RAISE
bus_system=#
```

Дополнительное задание

Функция:

```
CREATE OR REPLACE FUNCTION check_punch_sequence()  
RETURNS TRIGGER AS $$  
DECLARE  
    last_event BOOLEAN;  
BEGIN  
    SELECT is_out_punch INTO last_event  
    FROM time_punch  
    WHERE employee_id = NEW.employee_id  
    ORDER BY punch_time DESC, id DESC  
    LIMIT 1;  
    RAISE NOTICE 'Последнее событие: %', last_event;  
    RAISE NOTICE 'Новое событие: %', NEW.is_out_punch;  
    IF last_event IS NOT NULL THEN  
        IF last_event = NEW.is_out_punch THEN  
            RAISE EXCEPTION 'Ошибка: два события подряд одного типа  
(вход/выход) запрещены';  
        END IF;  
    ELSE  
        IF NEW.is_out_punch THEN  
            RAISE EXCEPTION 'Ошибка: первое событие не может быть  
выходом';  
        END IF;  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

Триггер:

```
CREATE TRIGGER trg_check_punch_sequence  
BEFORE INSERT ON time_punch FOR EACH ROW EXECUTE FUNCTION  
check_punch_sequence();
```

Проверка:

```
emp_time=# INSERT INTO time_punch (employee_id, is_out_punch, punch_time) VALUES (2, false, '2005-05-25 10:00:00');  
ЗАМЕЧАНИЕ: Последнее событие: <NULL>  
ЗАМЕЧАНИЕ: Новое событие: f  
INSERT 0 1  
emp_time=# INSERT INTO time_punch (employee_id, is_out_punch, punch_time) VALUES (2, false, '2005-05-25 10:02:00');  
ЗАМЕЧАНИЕ: Последнее событие: f  
ЗАМЕЧАНИЕ: Новое событие: f  
ОШИБКА: Ошибка: два события подряд одного типа (вход/выход) запрещены  
КОНТЕКСТ: функция PL/pgSQL check_punch_sequence(), строка 16, оператор RAISE
```

```
emp_time=# INSERT INTO time_punch (employee_id, is_out_punch, punch_time) VALUES (3, true, '2005-05-26 10:06:00');
ЗАМЕЧАНИЕ: Последнее событие: <NULL>
ЗАМЕЧАНИЕ: Новое событие: t
ОШИБКА: Ошибка: первое событие не может быть выходом
КОНТЕКСТ: функция PL/pgSQL check_punch_sequence(), строка 20, оператор RAISE
```

Выводы

При работе над лабораторной работой 5 я узнал, как создавать процедуры, функции и триггеры в базе данных PostgreSQL.