**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»**

Автор: Хурс П.И

Факультет: ИКТ

Группа: К3241

Преподаватель: Говорова М.М.



Санкт-Петербург 2023

**Оглавление**

[1. Создать процедуры согласно индивидуальному заданию 3](#_Toc152879395)

[2. Создать необходимый триггер 5](#_Toc152879396)

[Вывод 6](#_Toc152879397)

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

**Оборудование:** компьютерный класс.

**Программное обеспечение:** СУБД PostgreSQL, SQL Shell (psql).

**Практическое задание:**

1. Создать процедуры/функции 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.

# Создать процедуры согласно индивидуальному заданию

1. Для создания нового рейса на поезд.

CREATE OR REPLACE PROCEDURE create\_new\_train\_trip(

\_train\_id integer,

\_schedule\_id integer,

\_departure\_date date,

\_arrival\_date date,

\_train\_status varchar)

LANGUAGE plpgsql

AS $$

BEGIN

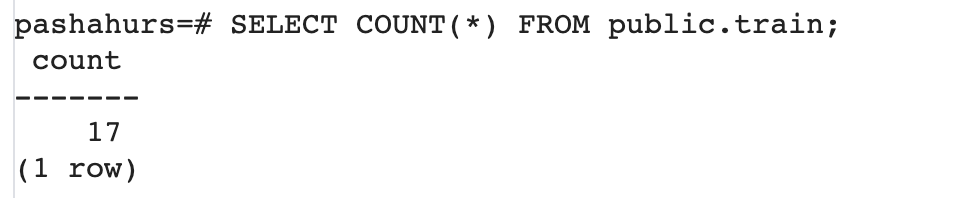
INSERT INTO public.train(train\_id, shedule\_id, departure\_date, arrival\_date, train\_status)

VALUES (\_train\_id, \_schedule\_id, \_departure\_date, \_arrival\_date, \_train\_status);

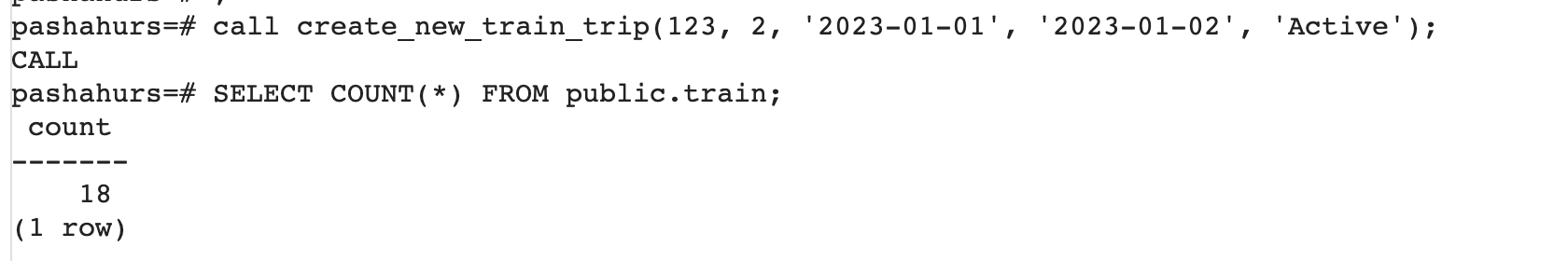
END;

$$;

До:



После:



1. Для формирования общей выручки по продаже билетов за сутки.

CREATE OR REPLACE FUNCTION total\_ticket\_sales\_revenue\_for\_day(sale\_date\_param date)

RETURNS bigint

LANGUAGE plpgsql AS

$$

DECLARE

total\_revenue bigint;

BEGIN

SELECT SUM(t.price) INTO total\_revenue

FROM public.ticket AS t

JOIN public.seat AS s ON t.seat\_id = s.seat\_id

JOIN public.carriage AS c ON s.carriage\_id = c.carriage\_id

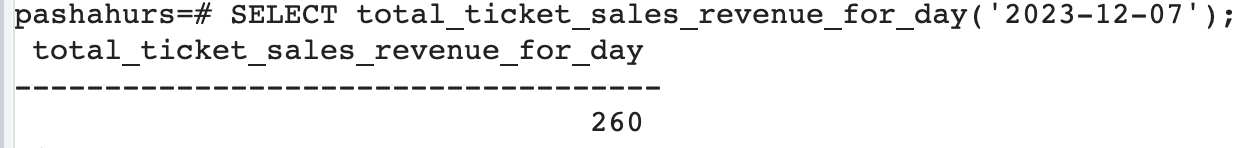
JOIN public.train AS tr ON c.train\_id = tr.train\_id

WHERE tr.departure\_date = sale\_date\_param;

RETURN total\_revenue;

END;

$$;



1. Для вычисления количества автомобилей заданной марки.

CREATE OR REPLACE PROCEDURE increase\_commuter\_train\_prices()

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE public.ticket

SET price = price \* 1.2

FROM public.seat, public.carriage, public.train, public.schedule

WHERE

public.ticket.seat\_id = public.seat.seat\_id AND

public.seat.carriage\_id = public.carriage.carriage\_id AND

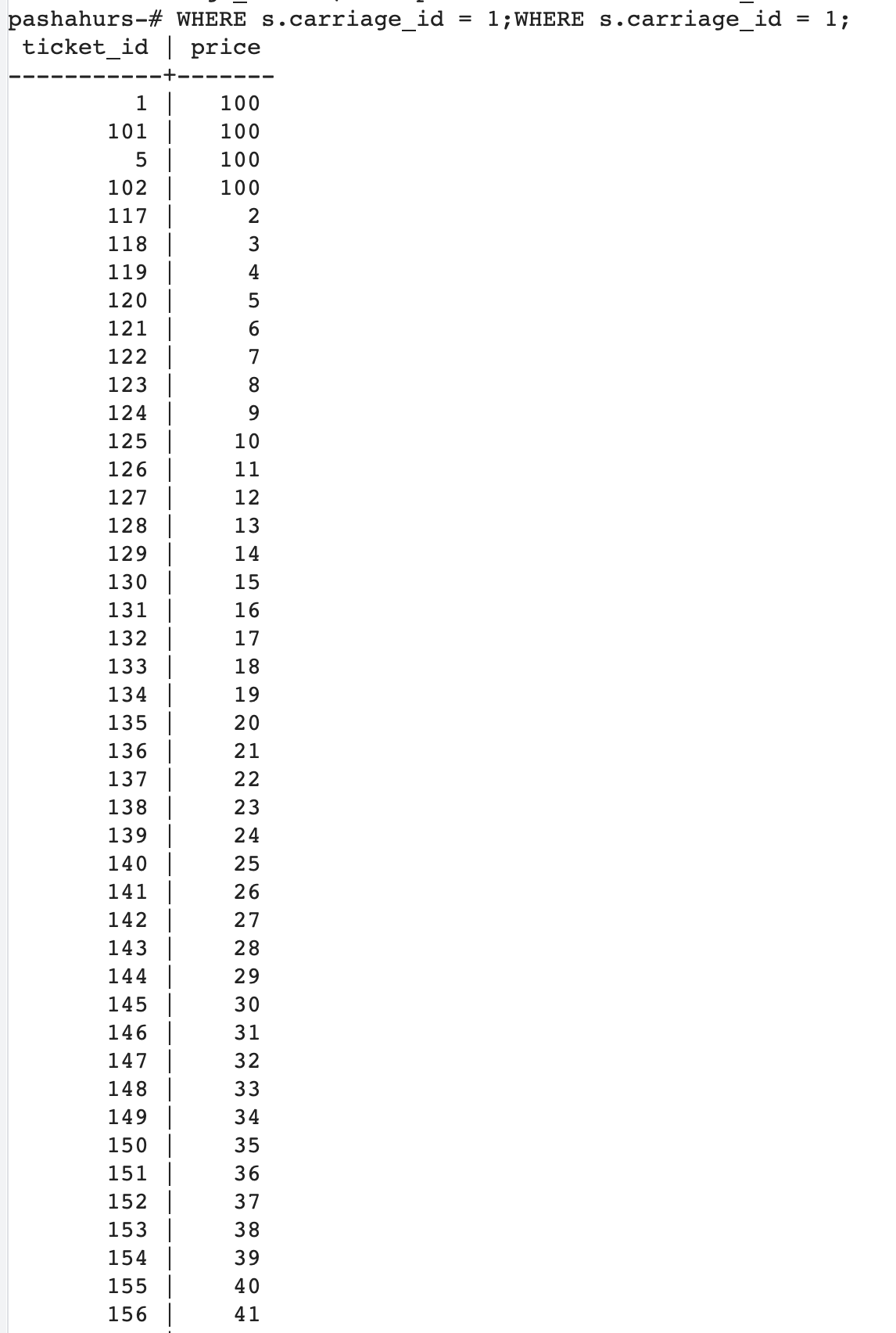
public.carriage.train\_id = public.train.train\_id AND

public.train.shedule\_id = public.schedule.schedule\_id AND

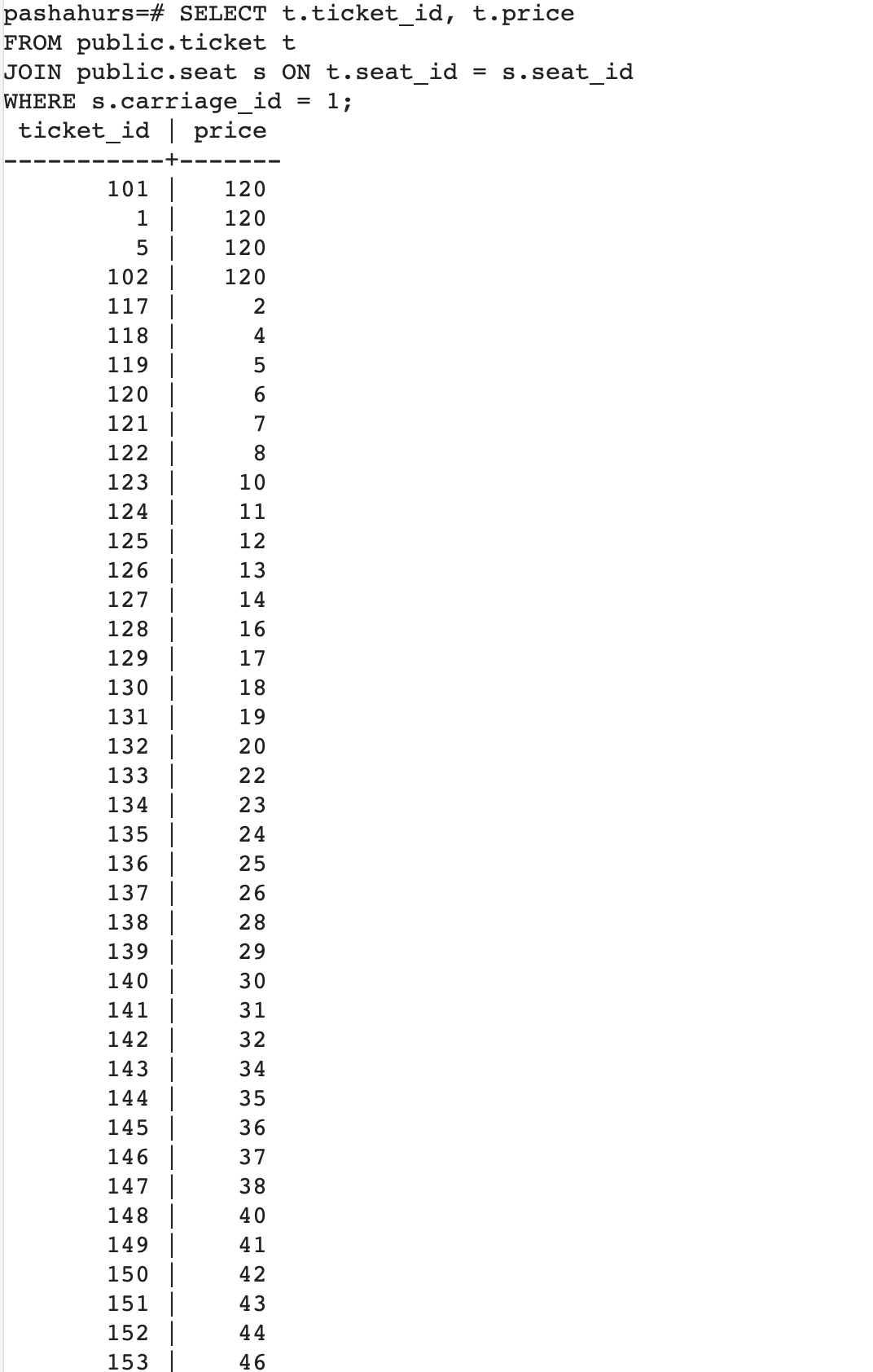
public.schedule.train\_type = 'Commuter';

END;

$$;



после



# Создать необходимый триггер

Триггер будет автоматически обновлять статус автомобиля на "недоступен" после того, как контракт на его аренду будет создан. Это логичное действие, поскольку автомобиль, который уже арендован, не должен быть доступен для новых аренд до возврата.

CREATE TABLE IF NOT EXISTS audit\_log (

log\_id SERIAL PRIMARY KEY,

table\_name VARCHAR(255),

action VARCHAR(50),

original\_data TEXT,

new\_data TEXT,

log\_timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE OR REPLACE FUNCTION universal\_audit\_trigger()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' THEN

INSERT INTO audit\_log (table\_name, action, new\_data)

VALUES (TG\_TABLE\_NAME, 'INSERT', row\_to\_json(NEW));

RETURN NEW;

ELSIF TG\_OP = 'DELETE' THEN

INSERT INTO audit\_log (table\_name, action, original\_data)

VALUES (TG\_TABLE\_NAME, 'DELETE', row\_to\_json(OLD));

RETURN OLD;

ELSIF TG\_OP = 'UPDATE' THEN

INSERT INTO audit\_log (table\_name, action, original\_data, new\_data)

VALUES (TG\_TABLE\_NAME, 'UPDATE', row\_to\_json(OLD), row\_to\_json(NEW));

RETURN NEW;

END IF;

RETURN NULL;

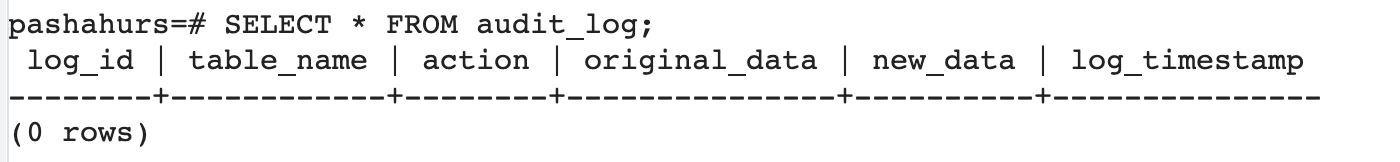
END;

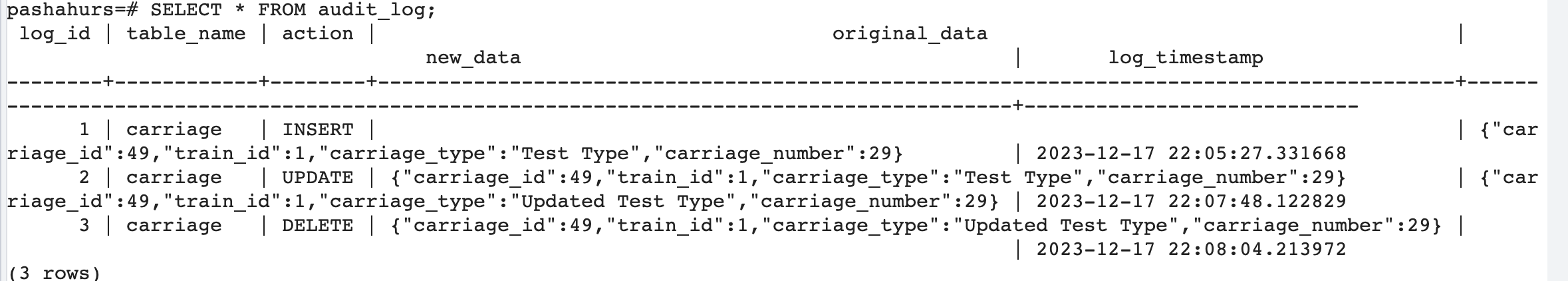
$$ LANGUAGE plpgsql;

CREATE TRIGGER carriage\_audit\_trigger

AFTER INSERT OR DELETE OR UPDATE ON carriage

FOR EACH ROW EXECUTE FUNCTION universal\_audit\_trigger();





# Вывод

В ходе лабораторной работы были успешно созданы и протестированы процедуры и триггеры в системе управления базами данных PostgreSQL. Работа позволила овладеть практическими навыками программирования в SQL и пониманием механизма триггеров.