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 Крамарь Кирилл Александрович
Факультет прикладной информатики
Группа K3239
Направление подготовки 09.03.03 Прикладная информатика
Образовательная программа Мобильные и сетевые технологии
2023 **Преподаватель** Говорова Марина Михайловна

Санкт-Петербург
2024/2025

СОДЕРЖАНИЕ

Стр.

1 Цель работы

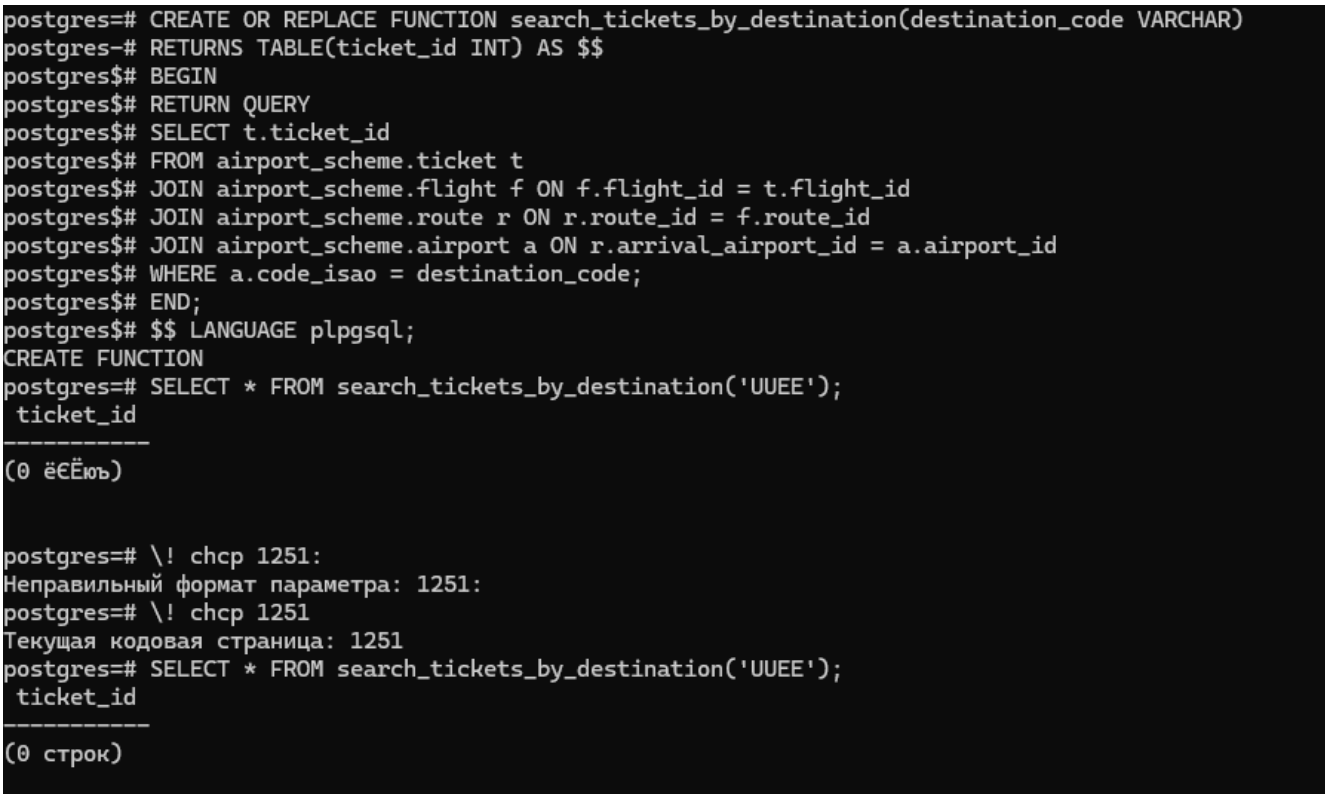
Цель работы: овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

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 ЛР5
3.1 Процедуры и функции

Процедура №1: Поиск билетов по пункту назначения
```
CREATE OR REPLACE FUNCTION
search_tickets_by_destination(destination_code VARCHAR)
RETURNS TABLE(ticket_id INT) AS $$
BEGIN
    RETURN QUERY
    SELECT t.ticket_id
    FROM airport_scheme.ticket t
    JOIN airport_scheme.flight f ON f.flight_id = t.flight_id
    JOIN airport_scheme.route r ON r.route_id = f.route_id
    JOIN airport_scheme.airport a ON r.arrival_airport_id = a.airport_id
    WHERE a.code_isao = destination_code;
END;
$$ LANGUAGE plpgsql;
```



Процедура №2: Создание новой кассы
```
CREATE OR REPLACE PROCEDURE add_cash_desk(
	online_status BOOLEAN,
	id INT,
	number INT
)
AS $$
BEGIN
	INSERT INTO airport_scheme.cash_desk(cashdesk_id, cashdesk_number, is_online)
	VALUES (id, number, online_status);
END;
$$ LANGUAGE plpgsql;
```

```
postgres=# CREATE OR REPLACE PROCEDURE add_cash_desk(
postgres(# online_status BOOLEAN,
postgres(# id INT,
postgres(# number INT
postgres(# )
postgres-# AS $$
postgres$# BEGIN
postgres$# INSERT INTO airport_scheme.cash_desk(cashdesk_id, cashdesk_number, is_online)
postgres$# VALUES (id, number, online_status);
postgres$# END;
postgres$# $$ LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL add_cash_desk(51, 141, TRUE)
postgres-#
```

Процедура №3: Определить расход топлива по всем маршрутам за истекший месяц.
```
CREATE OR REPLACE FUNCTION monthly_fuel_usage()
RETURNS TABLE(route_id INT, total_fuel NUMERIC)
AS $$
BEGIN
	RETURN QUERY
	SELECT
		r.route_id,
		SUM(ac.fuel_consumption * (
			EXTRACT(EPOCH FROM ((f.arrival_date + f.arrival_time) - (f.departure_date + f.departure_time))) / 3600
		)) AS total_fuel
	FROM airport_scheme.flight f
	JOIN airport_scheme.route r ON r.route_id = f.route_id
	JOIN airport_scheme.aircraft ac ON ac.aircraft_id = f.aircraft_id
	WHERE f.departure_date BETWEEN (CURRENT_DATE - INTERVAL '1
```

month') AND CURRENT_DATE
GROUP BY r.route_id;
END;
$$ LANGUAGE plpgsql;
```

```
postgres=# CREATE OR REPLACE FUNCTION monthly_fuel_usage()
postgres-# RETURNS TABLE(route_id INT, total_fuel NUMERIC)
postgres-# AS $$
postgres$# BEGIN
postgres$# RETURN QUERY
postgres$# SELECT
postgres$# r.route_id,
postgres$# SUM(ac.fuel_consumption * (
postgres$# EXTRACT(EPOCH FROM ((f.arrival_date + f.arrival_time) - (f.departure_date + f.departure_tim
e)))
postgres$# / 3600
postgres$# )) AS total_fuel
postgres$# FROM airport_scheme.flight f
postgres$# JOIN airport_scheme.route r ON r.route_id = f.route_id
postgres$# JOIN airport_scheme.aircraft ac ON ac.aircraft_id = f.aircraft_id
postgres$# WHERE f.departure_date BETWEEN (CURRENT_DATE - INTERVAL '1 month') AND CURRENT_DATE
postgres$# GROUP BY r.route_id;
postgres$# END;
postgres$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT * FROM monthly_fuel_usage();
```

route_id	fuel_consumption
29	10.6000000000000000000
34	9.2000000000000000000
32	10.2000000000000000000
10	9.2000000000000000000
9	10.0000000000000000000
7	10.6000000000000000000
35	10.0000000000000000000
15	10.2000000000000000000
6	8.4000000000000000000
```

3.2 Триггеры, Вариант 2.1

Триггер №1: Лог изменений в таблице cash_desk

```
CREATE TABLE IF NOT EXISTS airport_scheme.cash_desk_log (
	log_id SERIAL PRIMARY KEY,
	action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
	action_type TEXT,
	cashdesk_id INT,
	old_status BOOLEAN,
	new_status BOOLEAN
);
```

```
postgres=# CREATE TABLE IF NOT EXISTS airport_scheme.cash_desk_log (
postgres(# log_id SERIAL PRIMARY KEY,
postgres(# action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
postgres(# action_type TEXT,
postgres(# cashdesk_id INT,
postgres(# old_status BOOLEAN,
postgres(# new_status BOOLEAN
postgres(# );
CREATE TABLE
```

```
CREATE OR REPLACE FUNCTION log_cash_desk_changes()
RETURNS TRIGGER AS $$
BEGIN
	IF TG_OP = 'UPDATE' THEN
		INSERT INTO airport_scheme.cash_desk_log(action_type, cashdesk_id,
old_status, new_status)
		VALUES ('UPDATE', NEW.cashdesk_id, OLD.is_online,
NEW.is_online);
	END IF;
	RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_cash_desk_update
AFTER UPDATE ON airport_scheme.cash_desk
FOR EACH ROW EXECUTE FUNCTION log_cash_desk_changes();
```

```
postgres=# CREATE TABLE IF NOT EXISTS airport_scheme.cash_desk_log (
postgres(# log_id SERIAL PRIMARY KEY,
postgres(# action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
postgres(# action_type TEXT,
postgres(# cashdesk_id INT,
postgres(# old_status BOOLEAN,
postgres(# new_status BOOLEAN
postgres(# );
CREATE TABLE
postgres=# CREATE OR REPLACE FUNCTION log_cash_desk_changes()
postgres-# RETURNS TRIGGER AS $$
postgres$# BEGIN
postgres$# IF TG_OP = 'UPDATE' THEN
postgres$# INSERT INTO airport_scheme.cash_desk_log(action_type, cashdesk_id, old_status, new_statu
s)
postgres$# VALUES ('UPDATE', NEW.cashdesk_id, OLD.is_online, NEW.is_online);
postgres$# END IF;
postgres$# RETURN NEW;
postgres$# END;
postgres$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER trg_cash_desk_update
postgres-# AFTER UPDATE ON airport_scheme.cash_desk
postgres-# FOR EACH ROW EXECUTE FUNCTION log_cash_desk_changes();
CREATE TRIGGER
postgres=#
```

Триггер №2: Блокировать установку кассы с номером ниже 0

```
CREATE OR REPLACE FUNCTION check_cashdesk_number()
RETURNS TRIGGER AS $$
BEGIN
	IF NEW.cashdesk_number < 0 THEN
		RAISE EXCEPTION 'Номер кассы не может быть отрицательным';
	END IF;
	RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_cashdesk_number
BEFORE INSERT OR UPDATE ON airport_scheme.cash_desk
FOR EACH ROW EXECUTE FUNCTION check_cashdesk_number();
```

```
postgres=# CREATE OR REPLACE FUNCTION check_cashdesk_number()
postgres-# RETURNS TRIGGER AS $$
postgres$# BEGIN
postgres$# IF NEW.cashdesk_number < 0 THEN
postgres$# RAISE EXCEPTION 'Номер кассы не может быть отрицательным'
postgres$# END IF;
postgres$# RETURN NEW;
postgres$# END;
postgres$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER trg_check_cashdesk_number
postgres-# BEFORE INSERT OR UPDATE ON airport_scheme.cash_desk
postgres-# FOR EACH ROW EXECUTE FUNCTION check_cashdesk_number();
CREATE TRIGGER
postgres=#
```

Триггер №3: Логирование регистрации пассажиров

```
CREATE TABLE IF NOT EXISTS airport_scheme.passenger_log (
	log_id SERIAL PRIMARY KEY,
	passport_id INTEGER,
	registration_status BOOLEAN,
	changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE OR REPLACE FUNCTION log_passenger_registration()
RETURNS TRIGGER AS $$
BEGIN
	INSERT INTO airport_scheme.passenger_log(passport_id, registration_status)
	VALUES (NEW.passport_id, NEW.registration_status);
	RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_passenger_registration
AFTER INSERT OR UPDATE ON airport_scheme.passenger
FOR EACH ROW EXECUTE FUNCTION log_passenger_registration();
```

```
postgres=# CREATE TABLE IF NOT EXISTS airport_scheme.passenger_log (
postgres(# log_id SERIAL PRIMARY KEY,
postgres(# passport_id INTEGER,
postgres(# registration_status BOOLEAN,
postgres(# changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
postgres(# );
CREATE TABLE
postgres=# CREATE OR REPLACE FUNCTION log_passenger_registration()
postgres-# RETURNS TRIGGER AS $$
postgres$# BEGIN
postgres$# INSERT INTO airport_scheme.passenger_log(passport_id, registration_status)
postgres$# VALUES (NEW.passport_id, NEW.registration_status);
postgres$# RETURN NEW;
postgres$# END;
postgres$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER trg_passenger_registration
postgres-# AFTER INSERT OR UPDATE ON airport_scheme.passenger
postgres-# FOR EACH ROW EXECUTE FUNCTION log_passenger_registration();
CREATE TRIGGER
postgres=#
```

3.3 Триггеры, Вариант 2.2

Триггер №1: Проверка непустой должности у члена команды

```
CREATE OR REPLACE FUNCTION check_crew_role_not_empty()
RETURNS TRIGGER AS $$
BEGIN
	IF NEW.role IS NULL OR LENGTH(TRIM(NEW.role)) = 0 THEN
		RAISE EXCEPTION 'Роль экипажа не может быть пустой';
	END IF;
	RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_crew_role
BEFORE INSERT OR UPDATE ON airport_scheme.crew
FOR EACH ROW
EXECUTE FUNCTION check_crew_role_not_empty();
```

```
postgres=# CREATE OR REPLACE FUNCTION check_crew_role_not_empty()
postgres-# RETURNS TRIGGER AS $$
postgres$# BEGIN
postgres$# IF NEW.role IS NULL OR LENGTH(TRIM(NEW.role)) = 0 THEN
postgres$# RAISE EXCEPTION 'Роль экипажа не может быть пустой';
postgres$# END IF;
postgres$# RETURN NEW;
postgres$# END;
postgres$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER trg_check_crew_role
postgres-# BEFORE INSERT OR UPDATE ON airport_scheme.crew
postgres-# FOR EACH ROW
postgres-# EXECUTE FUNCTION check_crew_role_not_empty();
CREATE TRIGGER
postgres=#
```

Триггер №2: Автоматическая регистрация даты создания билета

```
ALTER TABLE airport_scheme.ticket ADD COLUMN IF NOT EXISTS created_at TIMESTAMP;

CREATE OR REPLACE FUNCTION auto_set_ticket_time()
RETURNS TRIGGER AS $$
BEGIN
	NEW.created_at := CURRENT_TIMESTAMP;
	RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_auto_set_ticket_time
BEFORE INSERT ON airport_scheme.ticket
FOR EACH ROW EXECUTE FUNCTION auto_set_ticket_time();
```

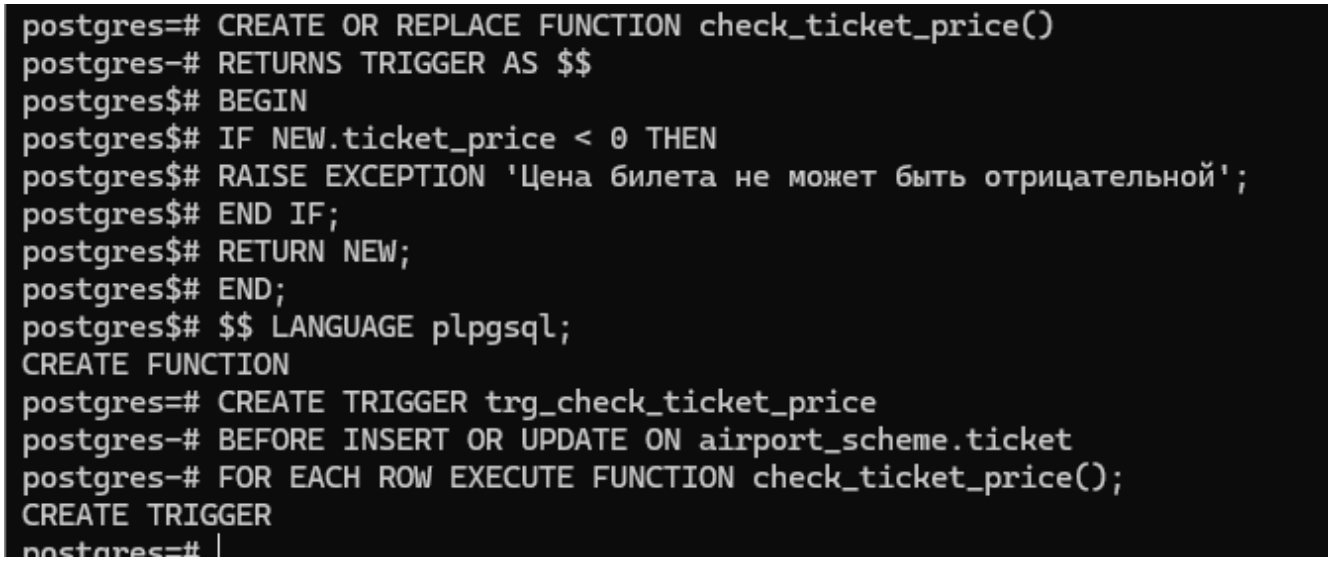
```
postgres=# ALTER TABLE airport_scheme.ticket ADD COLUMN IF NOT EXISTS created_at TIMESTAMP;
ALTER TABLE
postgres=# CREATE OR REPLACE FUNCTION auto_set_ticket_time()
postgres-# RETURNS TRIGGER AS $$
postgres$# BEGIN
postgres$# NEW.created_at := CURRENT_TIMESTAMP;
postgres$# RETURN NEW;
postgres$# END;
postgres$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER trg_auto_set_ticket_time
postgres-# BEFORE INSERT ON airport_scheme.ticket
postgres-# FOR EACH ROW EXECUTE FUNCTION auto_set_ticket_time();
CREATE TRIGGER
postgres=#
```

Триггер №3: Блокировка отрицательной стоимости билета

```
CREATE OR REPLACE FUNCTION check_ticket_price()
```

```
RETURNS TRIGGER AS $$
BEGIN
	IF NEW.ticket_price < 0 THEN
		RAISE EXCEPTION 'Цена билета не может быть отрицательной';
	END IF;
	RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_ticket_price
BEFORE INSERT OR UPDATE ON airport_scheme.ticket
FOR EACH ROW EXECUTE FUNCTION check_ticket_price();
```



Триггер №4: Автоматическая установка "даты последнего обслуживания" при добавлении нового самолета

```
CREATE OR REPLACE FUNCTION set_default_maintenance_date()
RETURNS TRIGGER AS $$
BEGIN
	IF NEW.last_maintenance_date IS NULL THEN
		NEW.last_maintenance_date := CURRENT_DATE;
	END IF;
	RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```
```

```
CREATE TRIGGER trg_set_maintenance_date
BEFORE INSERT ON airport_scheme.aircraft
EXECUTE FUNCTION set_default_maintenance_date();
```
```

```
postgres=# CREATE OR REPLACE FUNCTION set_default_maintenance_date()
postgres-# RETURNS TRIGGER AS $$
postgres$# BEGIN
postgres$# IF NEW.last_maintenance_date IS NULL THEN
postgres$# NEW.last_maintenance_date := CURRENT_DATE;
postgres$# END IF;
postgres$# RETURN NEW;
postgres$# END;
postgres$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER trg_set_maintenance_date
postgres-# BEFORE INSERT ON airport_scheme.aircraft
postgres-# EXECUTE FUNCTION set_default_maintenance_date();
CREATE TRIGGER
```

Выводы

В данной лабораторной работе были получены основные навыки создания функций, процедур и триггеров , а также навыки по работе в plpgsql.