МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра цифровой трансформации (ЦТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

по дисциплине «Разработка баз данных»

Практическое занятие №6

Студенты группы *ИКБО-66-23 Смирнов А.Ю.* _____
(подпись)

Ассистент *Копылова Я.А.*

_____
(подпись)

Отчет представлен «___»_____2025 г.

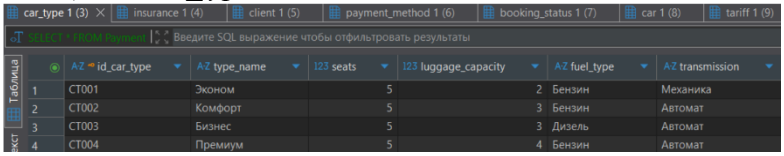
Москва 2025 г.

ПРАКТИЧЕСКАЯ РАБОТА №6. ТРИГГЕРЫ И КУРСОРЫ В POSTGRESQL

Цель работы:

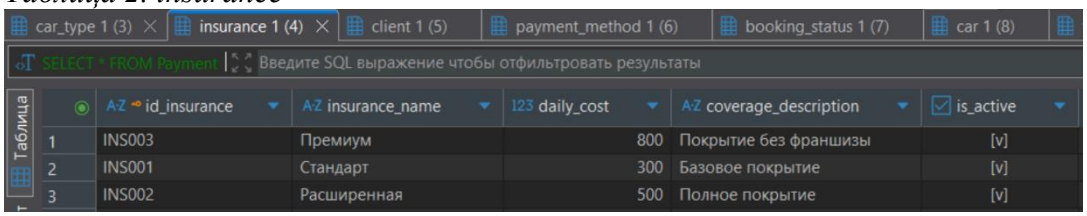
Целью данной практической работы является формирование у студентов углубленных практических навыков по управлению данными и реализации сложной бизнес-логики в СУБД PostgreSQL с использованием триггеров и курсоров.

Таблица 1. Car_type



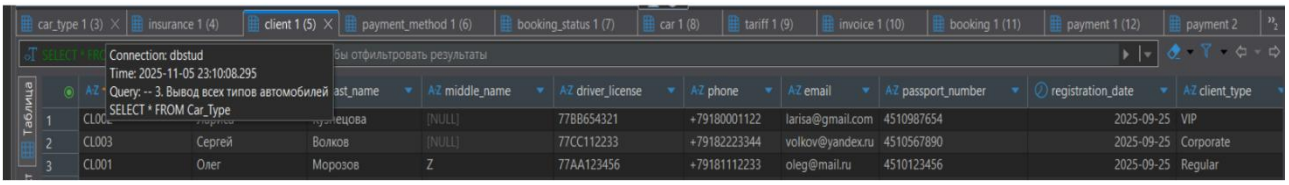
	id_car_type	type_name	seats	luggage_capacity	fuel_type	transmission
1	CT001	Эконом	5	2	Бензин	Механика
2	CT002	Комфорт	5	3	Бензин	Автомат
3	CT003	Бизнес	5	3	Дизель	Автомат
4	CT004	Премиум	5	4	Бензин	Автомат

Таблица 2. insurance



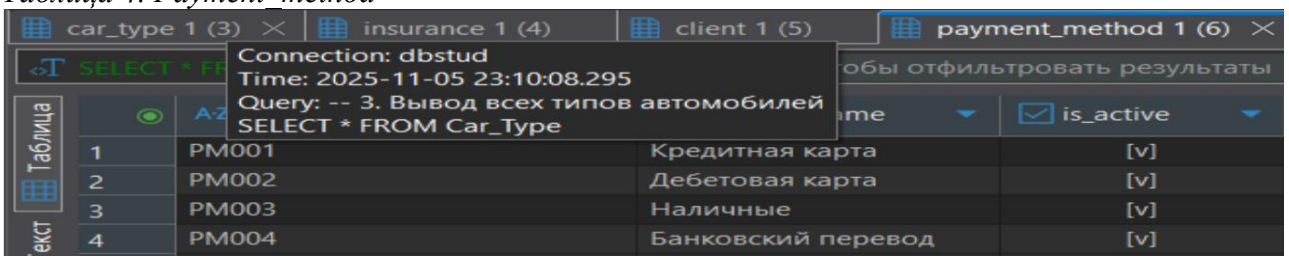
	id_insurance	insurance_name	daily_cost	coverage_description	is_active
1	INS003	Премиум	800	Покрытие без франшизы	[v]
2	INS001	Стандарт	300	Базовое покрытие	[v]
3	INS002	Расширенная	500	Полное покрытие	[v]

Таблица 3. Client



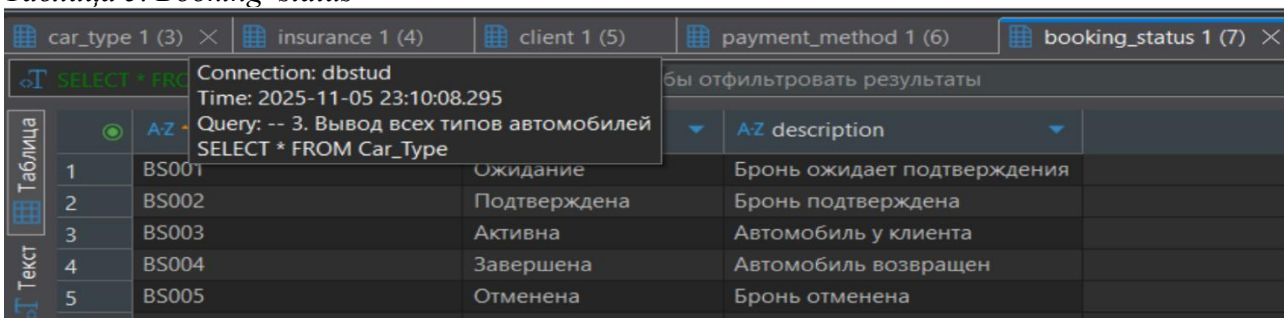
1	CL00		Кузнецова	[NULL]	77BB654321	+79180001122	larisa@gmail.com	4510987654	2025-09-25	VIP
2	CL003	Сергей	Волков	[NULL]	77CC112233	+79182223344	volkov@yandex.ru	4510567890	2025-09-25	Corporate
3	CL001	Олег	Морозов	Z	77AA123456	+79181112233	oleg@mail.ru	4510123456	2025-09-25	Regular

Таблица 4. Payment_method



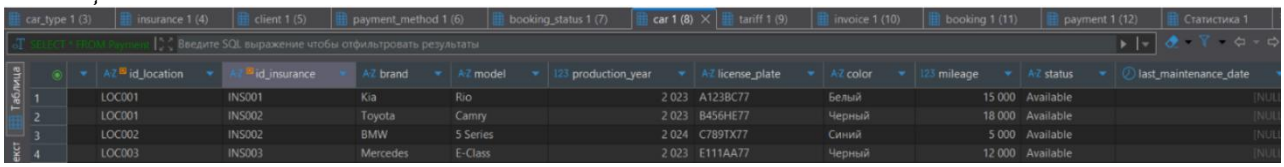
		is_active
1	PM001	[v]
2	PM002	[v]
3	PM003	[v]
4	PM004	[v]

Таблица 5. Booking_status



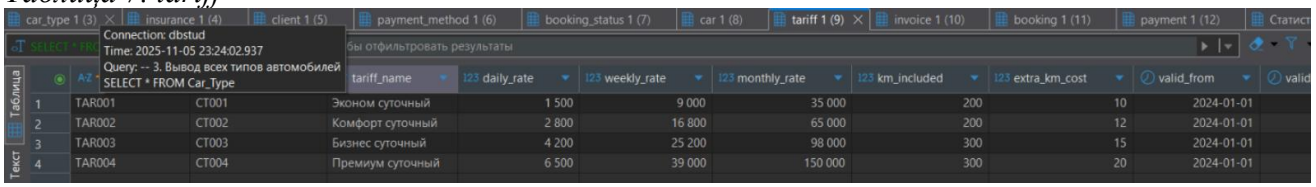
			description
1	BS001	Ожидание	Бронь ожидает подтверждения
2	BS002	Подтверждена	Бронь подтверждена
3	BS003	Активна	Автомобиль у клиента
4	BS004	Завершена	Автомобиль возвращен
5	BS005	Отменена	Бронь отменена

Таблица 6. Car



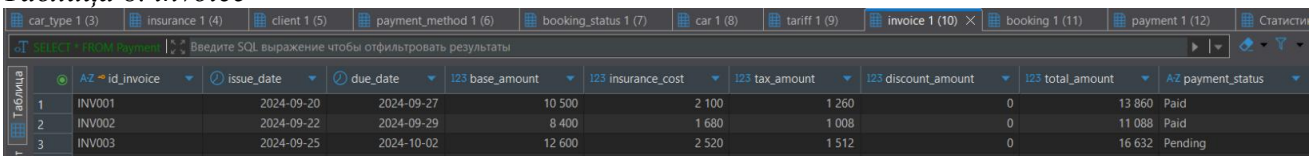
	id_location	id_insurance	brand	model	production_year	license_plate	color	mileage	status	last_maintenance_date
1	LOC001	INS001	Kia	Rio	2 023	A123BC77	Белый	15 000	Available	[NULL]
2	LOC001	INS002	Toyota	Camry	2 023	B456HE77	Черный	18 000	Available	[NULL]
3	LOC002	INS002	BMW	5 Series	2 024	C789TX77	Синий	5 000	Available	[NULL]
4	LOC003	INS003	Mercedes	E-Class	2 023	E111AA77	Черный	12 000	Available	[NULL]

Таблица 7. tariff



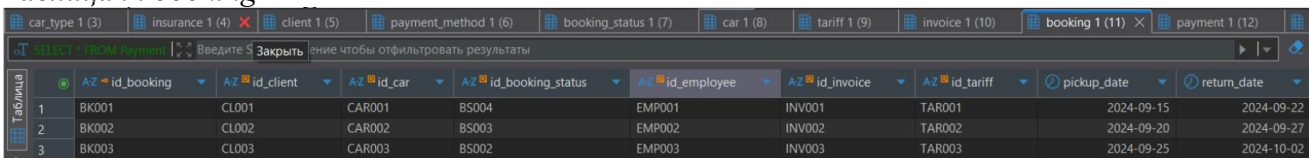
	id_tariff	id_car_type	tariff_name	daily_rate	weekly_rate	monthly_rate	km_included	extra_km_cost	valid_from	valid_to
1	TAR001	CT001	Эконом суточный	1 500	9 000	35 000	200	10	2024-01-01	[NULL]
2	TAR002	CT002	Комфорт суточный	2 800	16 800	65 000	200	12	2024-01-01	[NULL]
3	TAR003	CT003	Бизнес суточный	4 200	25 200	98 000	300	15	2024-01-01	[NULL]
4	TAR004	CT004	Премиум суточный	6 500	39 000	150 000	300	20	2024-01-01	[NULL]

Таблица 8. invoice



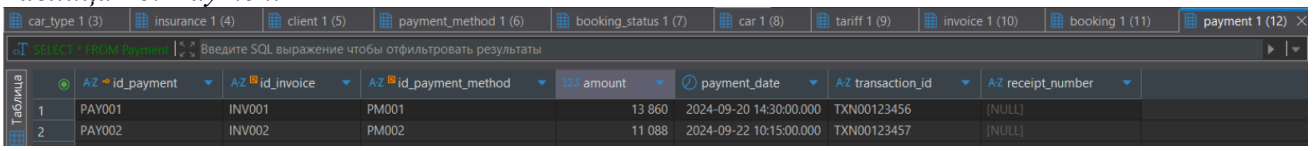
	id_invoice	issue_date	due_date	base_amount	insurance_cost	tax_amount	discount_amount	total_amount	payment_status
1	INV001	2024-09-20	2024-09-27	10 500	2 100	1 260	0	13 860	Paid
2	INV002	2024-09-22	2024-09-29	8 400	1 680	1 008	0	11 088	Paid
3	INV003	2024-09-25	2024-10-02	12 600	2 520	1 512	0	16 632	Pending

Таблица 9. booking



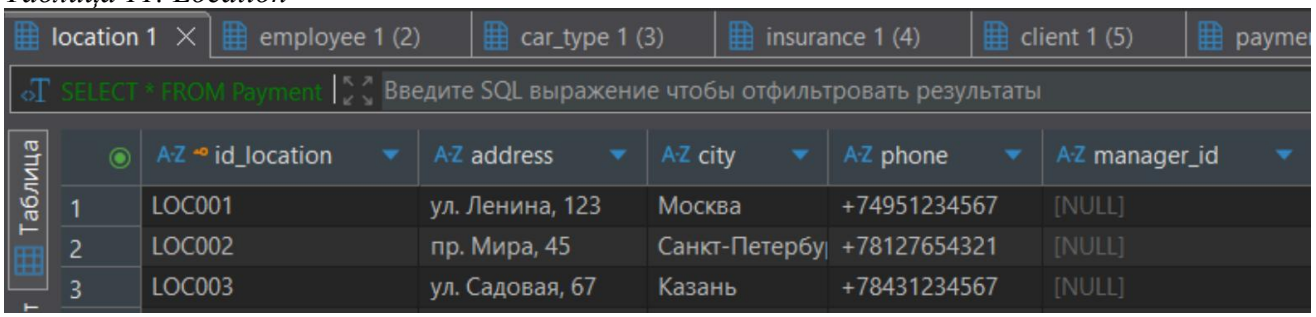
	id_booking	id_client	id_car	id_booking_status	id_employee	id_invoice	id_tariff	pickup_date	return_date
1	BK001	CL001	CAR001	BS004	EMP001	INV001	TAR001	2024-09-15	2024-09-22
2	BK002	CL002	CAR002	BS003	EMP002	INV002	TAR002	2024-09-20	2024-09-27
3	BK003	CL003	CAR003	BS002	EMP003	INV003	TAR003	2024-09-25	2024-10-02

Таблица 10. Payment



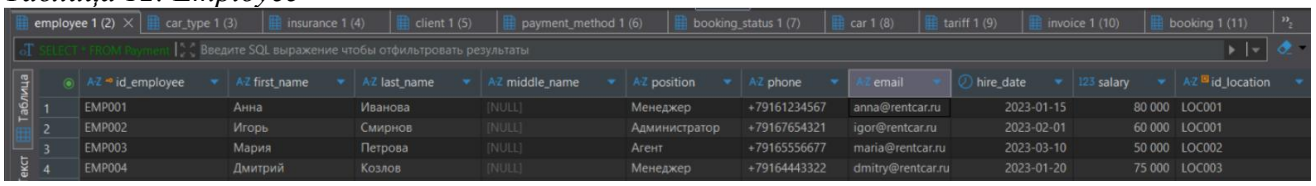
	id_payment	id_invoice	id_payment_method	amount	payment_date	transaction_id	receipt_number
1	PAY001	INV001	PM001	13 860	2024-09-20 14:30:00.000	TXN00123456	[NULL]
2	PAY002	INV002	PM002	11 088	2024-09-22 10:15:00.000	TXN00123457	[NULL]

Таблица 11. Location



	id_location	address	city	phone	manager_id
1	LOC001	ул. Ленина, 123	Москва	+74951234567	[NULL]
2	LOC002	пр. Мира, 45	Санкт-Петербу	+78127654321	[NULL]
3	LOC003	ул. Садовая, 67	Казань	+78431234567	[NULL]

Таблица 12. Employee



	id_employee	first_name	last_name	middle_name	position	phone	email	hire_date	salary	id_location
1	EMP001	Анна	Иванова	[NULL]	Менеджер	+79161234567	anna@rentcar.ru	2023-01-15	80 000	LOC001
2	EMP002	Игорь	Смирнов	[NULL]	Администратор	+79167654321	igor@rentcar.ru	2023-02-01	60 000	LOC001
3	EMP003	Мария	Петрова	[NULL]	Агент	+79165556677	maria@rentcar.ru	2023-03-10	50 000	LOC002
4	EMP004	Дмитрий	Козлов	[NULL]	Менеджер	+79164443322	dmitry@rentcar.ru	2023-01-20	75 000	LOC003

1.1 Триггер 1: сложная валидация

Листинг 1. ***Функция валидации, выполняемая ПЕРЕД вставкой в booking***

```
CREATE OR REPLACE FUNCTION validate_booking_insert()
RETURNS TRIGGER AS $$
DECLARE
    car_status VARCHAR(20);
BEGIN
    SELECT status INTO car_status
    FROM Car
    WHERE id_car = NEW.id_car
    FOR UPDATE; -- Защита от "состояния гонки"

    IF car_status != 'Available' THEN
        RAISE EXCEPTION 'Автомобиль недоступен для бронирования
(ID: %). Статус: %',
            NEW.id_car, car_status;
    END IF;

    UPDATE Car SET status = 'Booked' WHERE id_car = NEW.id_car;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

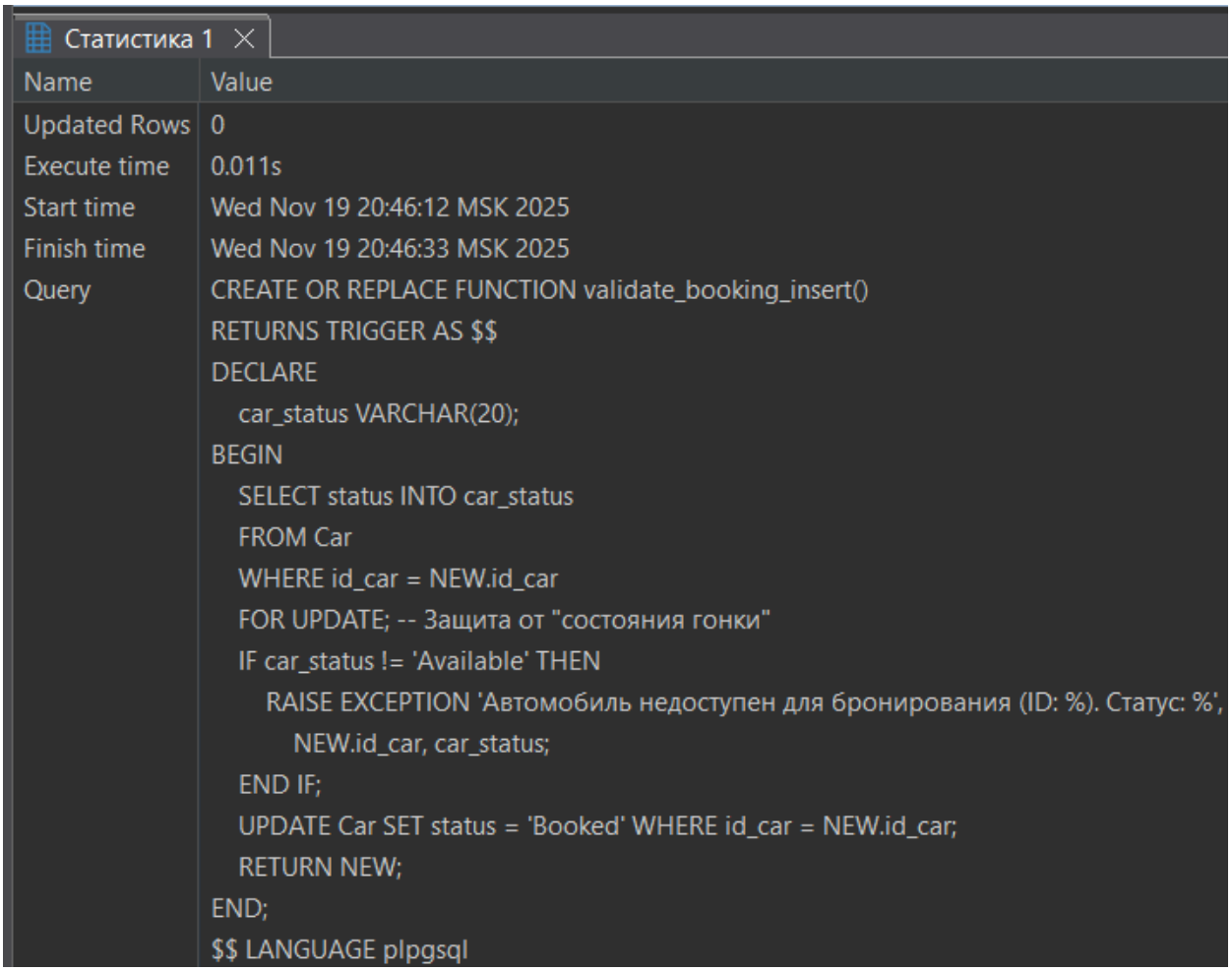


Рисунок 1 – Результат создания 1 триггерной функции

Далее нам необходимо привязать созданную нами триггерную функцию к конкретному событию, когда она будет срабатывать. Задаём все параметры, которые обговорили на этапе описания алгоритма.

Листинг 2. Привязка триггера BEFORE INSERT

```
CREATE TRIGGER check_booking_availability
    BEFORE INSERT ON Booking
    FOR EACH ROW
    EXECUTE FUNCTION validate_booking_insert();
```

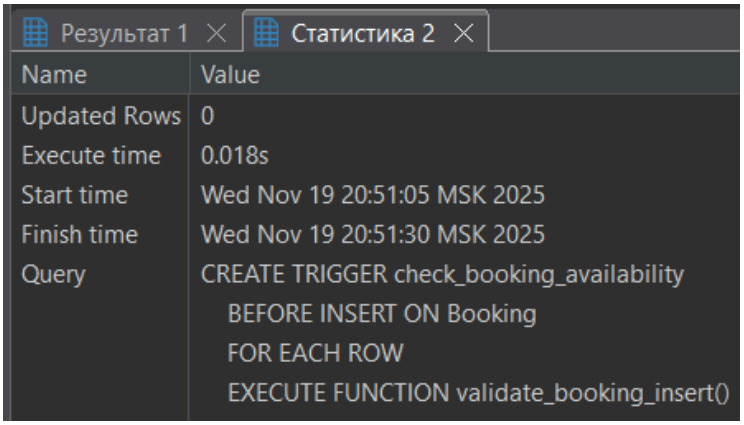


Рисунок 2 – Результат привязки 1 триггерной функции

1.1.1 Демонстрация (тестирование) работы триггера

Для теста используем данные автомобиля CAR001 (статус: 'Maintenance')

ТЕСТ 1: попытка забронировать автомобиль CAR001, когда он на обслуживании.

Ожидаемый результат: провал, так как автомобиль недоступен для бронирования.

Листинг 3. Запрос, который должен быть прерван триггером

```
INSERT INTO Booking (id_booking, id_client, id_car,
id_booking_status, id_employee, id_invoice, id_tariff, pickup_date,
return_date, pickup_location, return_location)
VALUES ('BK004', 'CL001', 'CAR001', 'BS002', 'EMP001', 'INV001',
'TAR001', '2024-10-01', '2024-10-05', 'LOC001', 'LOC001');
```

Полученный результат полностью совпадает с ожидаемым – Рисунок 3.

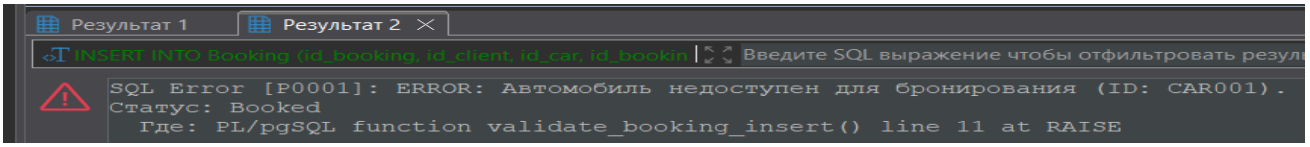


Рисунок 3 – Результаты проверки

ТЕСТ 2: попытка забронировать автомобиль CAR002, который доступен.

Ожидаемый результат: успешное создание бронирования.

Листинг 4. Запрос, который должен выполниться успешно

```
INSERT INTO Booking (id_booking, id_client, id_car,
id_booking_status, id_employee, id_invoice, id_tariff, pickup_date,
return_date, pickup_location, return_location)
VALUES ('BK007', 'CL002', 'CAR002', 'BS002', 'EMP002', 'INV002',
'TAR002', '2024-10-02', '2024-10-06', 'LOC001', 'LOC001');
```

Полученный результат снова полностью совпадает с ожидаемым, как видно по **Updated Rows**, которое говорит о том, сколько было изменено строк – Рисунок 4.

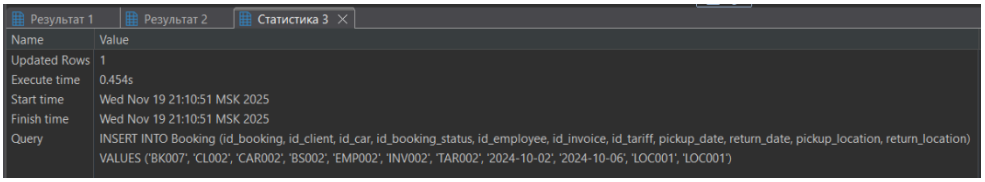


Рисунок 4 – Результаты проверки

Триггер был успешно реализован и прошёл необходимые проверки.

1.2 поддержание согласованности

Листинг 5. Функция, выполняемая ПОСЛЕ вставки/обновления/удаления

```
CREATE OR REPLACE FUNCTION update_invoice_after_booking_change()
RETURNS TRIGGER AS $$
DECLARE
    booking_total DECIMAL(10, 2);
    target_invoice_id VARCHAR(10);
BEGIN
    IF (TG_OP = 'DELETE') THEN
        target_invoice_id := OLD.id_invoice;
        UPDATE Car SET status = 'Available' WHERE id_car =
OLD.id_car;
    ELSE
        target_invoice_id := NEW.id_invoice;
        IF (TG_OP = 'INSERT') THEN
            UPDATE Car SET status = 'Booked' WHERE id_car =
NEW.id_car;
        ELSIF (TG_OP = 'UPDATE') THEN
            UPDATE Car SET status = 'Available' WHERE id_car =
OLD.id_car;
            UPDATE Car SET status = 'Booked' WHERE id_car =
NEW.id_car;
        END IF;
    END IF
```

```
SELECT COALESCE(SUM(t.daily_rate * b.total_days), 0)
    INTO booking_total
    FROM Booking b
    JOIN Tariff t ON b.id_tariff = t.id_tariff
    WHERE b.id_invoice = target_invoice_id;

    UPDATE Invoice
    SET total_amount = booking_total
    WHERE id_invoice = target_invoice_id;

    IF (TG_OP = 'DELETE') THEN
        RETURN OLD;
    ELSE
        RETURN NEW;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Статистика 1

Name	Value
Updated Rows	0
Execute time	0.020s
Start time	Wed Nov 19 21:31:17 MSK 2025
Finish time	Wed Nov 19 21:31:19 MSK 2025

Query	CREATE OR REPLACE FUNCTION update_invoice_after_booking_change()
	RETURNS TRIGGER AS $$
	DECLARE
	booking_total DECIMAL(10, 2);
	target_invoice_id VARCHAR(10);
	BEGIN
	IF (TG_OP = 'DELETE') THEN
	target_invoice_id := OLD.id_invoice;
	UPDATE Car SET status = 'Available' WHERE id_car = OLD.id_car;
	ELSE
	target_invoice_id := NEW.id_invoice;
	IF (TG_OP = 'INSERT') THEN
	UPDATE Car SET status = 'Booked' WHERE id_car = NEW.id_car;
	ELSIF (TG_OP = 'UPDATE') THEN
	UPDATE Car SET status = 'Available' WHERE id_car = OLD.id_car;
	UPDATE Car SET status = 'Booked' WHERE id_car = NEW.id_car;
	END IF;
	END IF;
	SELECT COALESCE(SUM(t.daily_rate * b.total_days), 0)
	INTO booking_total
	FROM Booking b
	JOIN Tariff t ON b.id_tariff = t.id_tariff
	WHERE b.id_invoice = target_invoice_id;
	UPDATE Invoice
	SET total_amount = booking_total
	WHERE id_invoice = target_invoice_id;
	IF (TG_OP = 'DELETE') THEN
	RETURN OLD;
	ELSE
	RETURN NEW;
	END IF;
	END;
	$$ LANGUAGE plpgsql

Рисунок 5 – Результат создания 2 триггерной функции

Далее нам снова необходимо привязать созданную нами триггерную функцию к конкретному событию, когда она будет срабатывать. Задаём все параметры, которые обговорили на этапе описания алгоритма.

Листинг 6. Привязка триггера AFTER

```
CREATE TRIGGER trg_update_invoice_after_booking
    AFTER INSERT OR UPDATE OR DELETE ON Booking
    FOR EACH ROW
    EXECUTE FUNCTION update_invoice_after_booking_change();
```

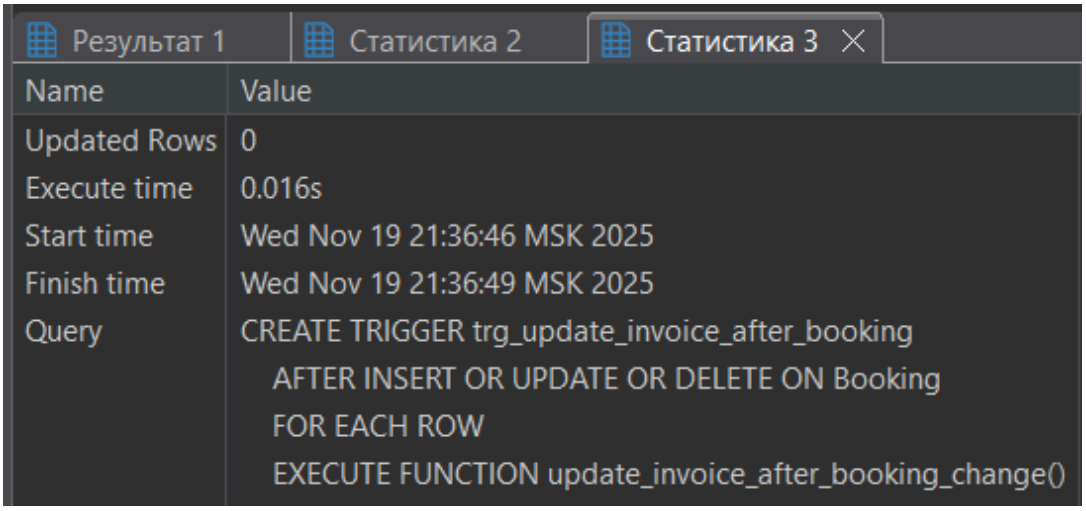


Рисунок 6 – Результат привязки 2 триггерной функции

1.2.1 Демонстрация (тестирование) работы триггера

Тест 1: **выполняем INSERT, добавляя новое бронирование CAR003. Состояние «до» (для INSERT):**

- Автомобиль CAR003: status = 'Available'
- Счет INV001: total_amount = 13860.00

Ожидаемый результат:

- Задача менеджера авто: занять автомобиль (статус должен стать 'Booked')
- пересчитать сумму счета (Сумма должна увеличиться с учетом нового бронирования).

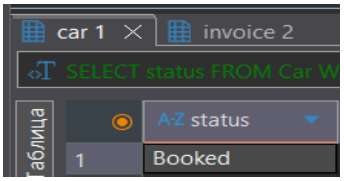
Листинг 7. Активация триггеров (INSERT)

```
-- Добавляем бронирование
INSERT INTO Booking (id_booking, id_client, id_car,
id_booking_status, id_employee, id_invoice, id_tariff, pickup_date,
return_date, pickup_location, return_location)
VALUES ('BK008', 'CL001', 'CAR003', 'BS002', 'EMP001', 'INV001',
'TAR003', '2024-10-03', '2024-10-08', 'LOC002', 'LOC002');

SELECT status FROM Car WHERE id_car = 'CAR003';

SELECT total_amount FROM Invoice WHERE id_invoice = 'INV001';
```

Полученный результат полностью совпадает с ожидаемым – Рисунок 3.



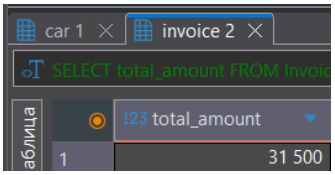


Рисунок 7 – Результаты проверки

ТЕСТ 2: вып**выполняем DELETE, удаляя бронирование BK008**.

Состояние «до» (для **UPDATE**):

- Автомобиль CAR003: status = 'Booked'.
- Счет INV001: total_amount = 31500.00

Ожидаемый результат:

- **Задача менеджера авто**: **вернуть автомобиль в доступные (статус должен стать 'Available')**.
- Задача **Бухгалтера**: **пересчитать сумму счета (Сумма должна вернуться к исходной 10500.00)**.

Листинг 8. Запрос, который должен выполниться успешно

```
-- Удаляем бронирование
DELETE FROM Booking WHERE id_booking = 'BK008';

-- Проверяем статус автомобиля
SELECT status FROM Car WHERE id_car = 'CAR003';

-- Проверяем сумму счета
SELECT total_amount FROM Invoice WHERE id_invoice = 'INV001';
```

Полученный результат снова полностью совпадает с ожидаемым, тест успешно пройден.

Рисунок 8 – Результаты проверки

1.3 Триггер 3: аудит и логирование

1.3.1 Код триггерной функции и триггера

Листинг 9. ***Функция логирования изменений статуса автомобиля***

```
CREATE OR REPLACE FUNCTION log_car_status_change()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.status IS DISTINCT FROM OLD.status THEN
        INSERT INTO car_audit
        (car_id, old_status, new_status, change_date, employee_id)
        VALUES
        (OLD.id_car, OLD.status, NEW.status, NOW(), 'SYS_AUTO');
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

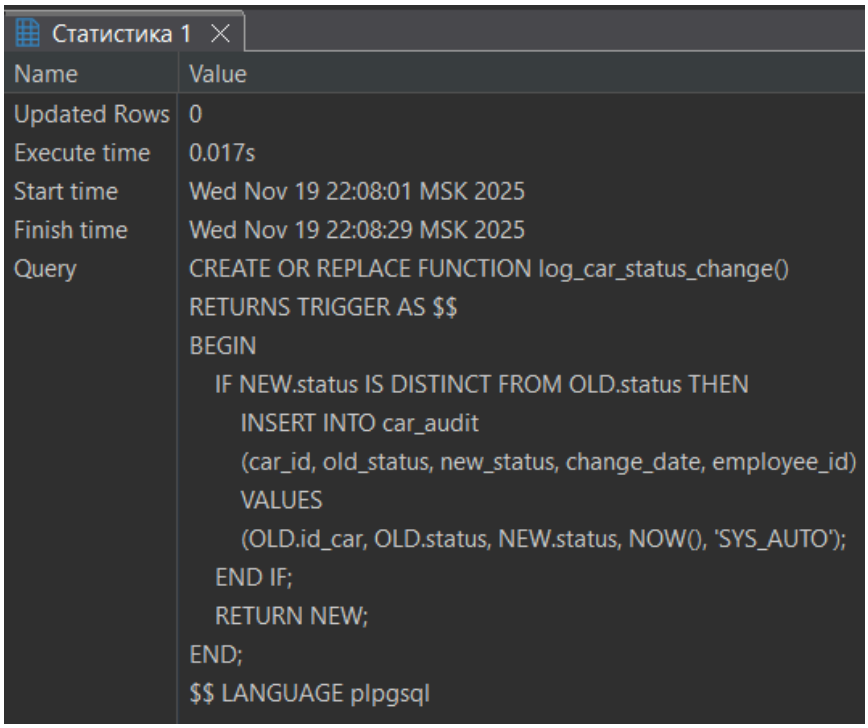


Рисунок 9 – Результат создания 3 триггерной функции

Далее нам необходимо привязать созданную нами триггерную функцию к конкретному событию, когда она будет срабатывать. Задаём все параметры, которые обговорили на этапе описания алгоритма.

Листинг 10. Привязка триггера AFTER UPDATE

```
create or replace trigger trg_log_car_status_change
    AFTER UPDATE ON Car
    FOR EACH ROW
    EXECUTE FUNCTION log_car_status_change();
```

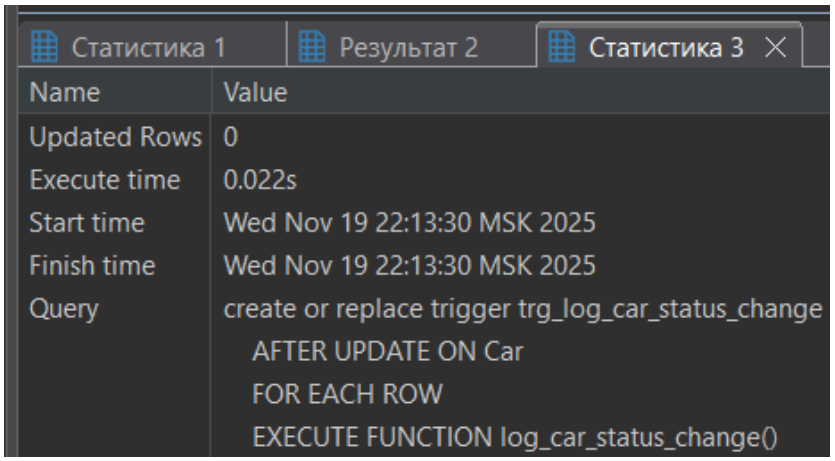


Рисунок 10 – Результат привязки 1 триггерной функции

1.3.2 Демонстрация (тестирование) работы триггера

Состояние «до»:

- В Car есть автомобиль CAR001 с статусом 'Available'.
- В car_audit 0 записей.

ТЕСТ 1: меняем цвет автомобиля CAR001.

Ожидаемый результат: триггер **не должен** сработать.

Листинг 11. Запрос, который должен быть прерван триггером

```
UPDATE Car
SET color = 'Красный'
WHERE id_car = 'CAR001';

SELECT COUNT(*) FROM car_audit;
```

Полученный результат полностью совпадает с ожидаемым. Количество строк не изменилось.

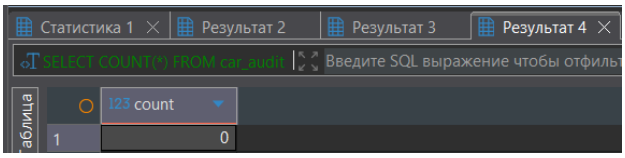


Рисунок 11 – Результаты проверки

ТЕСТ 2: меняем номер телефона клиента.

Ожидаемый результат: триггер **должен** сработать.

Листинг 12. Запрос, который должен выполниться успешно

```
UPDATE Car

SET status = 'Cleaning'

WHERE id_car = 'CAR001';



SELECT COUNT(*) FROM car_audit;
```

Полученный результат снова полностью совпадает с ожидаемым, теперь количество записей равно 1.

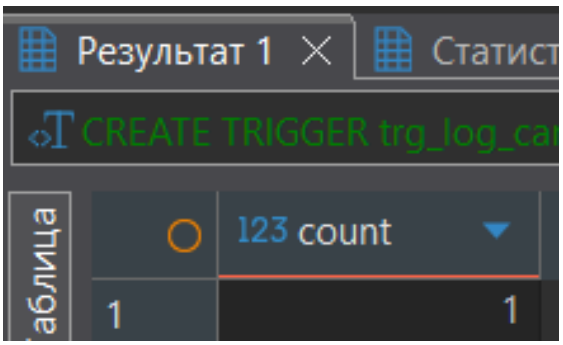


Рисунок 12 – Результаты проверки

Триггер был успешно реализован и прошёл необходимые проверки.

2. ИСПОЛЬЗОВАНИЕ КУРСОРОВ

- *Листинг 13. Пример явного курсора*

```
DO $$
DECLARE
    curs1 CURSOR FOR SELECT id_car, brand, model, status FROM Car;
    row_var RECORD;
BEGIN
    OPEN curs1;
    LOOP
        FETCH curs1 INTO row_var;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'ID: %, Марка: %, Модель: %, Статус: %',
                     row_var.id_car, row_var.brand, row_var.model,
row_var.status;
    END LOOP;
    CLOSE curs1;
```

Реализуем вышеописанный алгоритм, и смотрим на результат (он будет выведен в консоли).

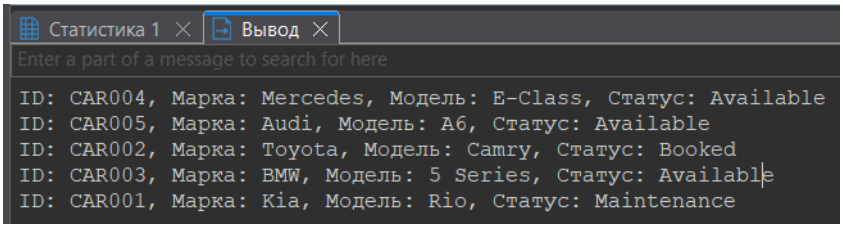


Рисунок 13 – Результат запроса

2.1 Неявный курсор в функции – вывод результата в таблицу

Листинг 14. Создание функции, возвращающей таблицу (с неявным курсором)

```
CREATE OR REPLACE FUNCTION get_all_cars_formatted()
RETURNS TABLE(car_id_out VARCHAR(10), car_info TEXT, status_out
VARCHAR(20))
AS $$
DECLARE
    car_row Car%ROWTYPE;
BEGIN
    FOR car_row IN
        SELECT * FROM Car ORDER BY id_car
    LOOP
        car_id_out := car_row.id_car;
        car_info := car_row.brand || ' ' || car_row.model || ' ('
|| car_row.color || ')';
        status_out := car_row.status;

        RETURN NEXT;
    END LOOP;

    RETURN;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM get_all_cars_formatted();
```

Ожидаемый результат: в отличие от Листинга 10, этот **SELECT** должен вернуть полноценную таблицу (во вкладке «Data» / «Result»).

	A-Z car_id_out	A-Z car_info	A-Z status_out
1	CAR001	Kia Rio (Красный)	Maintenance
2	CAR002	Toyota Camry (Черн	Booked
3	CAR003	BMW 5 Series (Сини	Available
4	CAR004	Mercedes E-Class (Ч	Available
5	CAR005	Audi A6 (Серый)	Available

Рисунок 14 – Результат запроса

И как мы видим, именно этот результат мы и получаем. Теперь мы сможем использовать этот результат либо напрямую, либо в других запросах.

1. ROW-level триггер выполняется для каждой строки, STATEMENT-level - один раз для всей операции. Пример: при массовом обновлении статусов автомобилей нужно логировать только факт операции, а не каждую строку.

2. При RETURN NULL в BEFORE-триггере операция пропускается для текущей строки. При RETURN OLD - UPDATE не изменяет строку (но не отменяет всю операцию).

3. RAISE NOTICE выводит сообщения в консоль, RETURN NEXT формирует строку в результирующей таблице. RAISE NOTICE не может возвращать структурированные данные.

4. TG_OP: 'INSERT', 'UPDATE', 'DELETE', 'TRUNCATE'. Определяет тип операции, вызвавшей триггер.

5. RAISE EXCEPTION полностью отменяет транзакцию с ошибкой, RETURN NULL только пропускает текущую строку. Для пользователя разница в том, что при EXCEPTION все изменения откатываются, а при NULL - только проблемная строка игнорируется.