

**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 Корольков Артем Алексеевич

Факультет прикладной информатики

Группа К3240

Направление подготовки 09.03.03 Прикладная информатика

Образовательная программа Мобильные и сетевые технологии 2023

Преподаватель Говорова Марина Михайловна

Санкт-Петербург

2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Выполнение	3
1.1 Название создаваемой базы данных	3
1.2 Схема логической модели базы данных, сгенерированная в Generate ERD	4
1.3 Создание хранимых процедур	5
1.3.1 Процедура для увеличения цены всех номеров на 5 %, если в отеле нет свободных номеров	5
1.3.2 Процедура для получения информации о свободных одноместных номерах отеля на завтрашний день	7
1.3.3 Процедура бронирования двухместного номера в гостинице на заданную дату и количество дней проживания	9
1.4 Создание триггеров	10
1.4.1 Триггер для автоматического обновления статуса номера при бронировании	10
1.4.2 Триггер для проверки дат бронирования	11
1.4.3 Триггер для автоматической очистки номера после выезда	12
1.4.4 Триггер для проверки минимальной стоимости бронирования	13
1.4.5 Триггер для автоматического создания уборки после освобождения номера	14
1.4.6 Триггер для проверки возраста гостя	15
1.4.7 Триггер для автоматического применения акций к бронированию	16
ЗАКЛЮЧЕНИЕ	18
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	19

ВВЕДЕНИЕ

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 ЛР2). 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).

1 Выполнение

1.1 Название создаваемой базы данных

Название базы данных – «Отель».

Изучение предметной области:

База данных разрабатывается для сети отелей, расположенных в различных городах.

Основная задача системы — хранение данных об отелях и управление информацией о бронировании, регистрации и проживании гостей.

Система должна поддерживать актуальные данные о состоянии номеров, которые могут быть свободны, забронированы или заняты. При заезде гости регистрируются, и информация о регистрации хранится в системе на протяжении года, после чего данные архивируются.

1.3 Создание хранимых процедур

1.3.1 Процедура для увеличения цены всех номеров на 5 %, если в отеле нет свободных номеров

```
CREATE OR REPLACE PROCEDURE increase_prices_if_no_vacancies()
LANGUAGE plpgsql
AS $$
DECLARE
    hotel_with_no_vacancies INTEGER;
BEGIN
    -- Проверяем, есть ли отели, где все номера заняты
    SELECT COUNT(*) INTO hotel_with_no_vacancies
    FROM Hotel h
    WHERE NOT EXISTS (
        SELECT 1 FROM Room r
        WHERE r.branch_code = h.branch_code
        AND r.occupied_status = false
    );

    -- Если такие отели есть, увеличиваем цены на 5%
    IF hotel_with_no_vacancies > 0 THEN
        UPDATE Price
        SET amount = amount * 1.05
        WHERE CURRENT_DATE BETWEEN period_start AND period_end;

        RAISE NOTICE 'Цены увеличены на 5%% для всех номеров, так как найдены отели без свободных номеров';
    ELSE
        RAISE NOTICE 'Во всех отелях есть свободные номера, цены не изменены';
    END IF;
END;
$$;
```

Рисунок 2 – Скриншот реализации процедуры 1.

Проверка в PSQL:

```
hotel_db=# SELECT * FROM Price;
```

price_id	amount	period_start	period_end	room_type_id
1	3500	2023-01-01	2023-12-31	1
2	5000	2023-01-01	2023-12-31	2
3	8000	2023-01-01	2023-12-31	3
4	12000	2023-01-01	2023-12-31	4
5	20000	2023-01-01	2023-12-31	5
6	2500	2025-04-27	2026-04-28	6
7	4000	2025-04-27	2026-04-28	7
8	3500	2025-04-27	2026-04-28	8
9	19835.83	2024-12-19	2025-07-17	5
10	3767.97	2025-03-16	2025-06-01	10
11	28028.45	2024-08-31	2025-08-04	12

```

hotel_db=# SELECT h.branch_code, h.name, r.room_id, r.number, r.occupied_status
hotel_db=# FROM Hotel h
hotel_db=# JOIN Room r ON h.branch_code = r.branch_code;

```

branch_code	name	room_id	number	occupied_status
7	Тхтхёёх тш эшх	1	165	t
11	Тршъры	2	468	t
26	Тюёёхъ	3	562	t
18	Тхтр	4	350	t
17	Тцэ√щ	5	405	t
19	Тхёёёрыёэ√щ	6	122	t
16	ТРҫцэ√щ	7	151	f
13	Тёёёш	8	376	t
15	Тёшьёёёшщ	9	258	t
16	ТРҫцэ√щ	10	348	f

```

hotel_db=# UPDATE Room SET occupied_status = true WHERE branch_code = 1;
UPDATE 39

```

```

hotel_db=# CALL increase_prices_if_no_vacancies();

```

ЗАМЕЧАНИЕ: Цены увеличены на 5% для всех номеров, так как найдены отели без свободных номеров
CALL

```

hotel_db=# SELECT * FROM Price;

```

price_id	amount	period_start	period_end	room_type_id
1	3500	2023-01-01	2023-12-31	1
2	5000	2023-01-01	2023-12-31	2
3	8000	2023-01-01	2023-12-31	3
4	12000	2023-01-01	2023-12-31	4
5	20000	2023-01-01	2023-12-31	5
12	11793.6	2024-12-30	2025-04-10	1
13	49196.47	2024-07-18	2024-10-12	2
16	33025.12	2024-10-31	2025-01-11	3
18	14406.63	2024-10-01	2025-02-19	3
20	24492.71	2024-09-19	2025-04-10	13

Рисунок 3 – Скриншоты проверки процедуры 1.

1.3.2 Процедура для получения информации о свободных одноместных номерах отеля на завтрашний день

```
CREATE OR REPLACE PROCEDURE get_available_single_rooms(  
    IN p_hotel_id INTEGER  
)  
LANGUAGE plpgsql  
AS $$  
DECLARE  
    r_record RECORD;  
BEGIN  
    -- Создаем временную таблицу для результатов  
    DROP TABLE IF EXISTS temp_available_rooms;  
    CREATE TEMP TABLE temp_available_rooms AS  
    SELECT  
        r.number AS room_number,  
        rt.name AS room_type,  
        p.amount AS base_price  
    FROM Room r  
    JOIN Room_type rt ON r.room_type_id = rt.room_type_id  
    JOIN Price p ON rt.room_type_id = p.room_type_id  
    WHERE r.branch_code = p_hotel_id  
        AND rt.beds_count = 1  
        AND r.occupied_status = false  
        AND r.cleaning_status = true  
        AND (CURRENT_DATE + INTERVAL '1 day') BETWEEN p.period_start AND p.period_end  
        AND NOT EXISTS (  
            SELECT 1 FROM Booking b  
            WHERE b.room_id = r.room_id  
            AND (CURRENT_DATE + INTERVAL '1 day') BETWEEN b.check_in_date AND b.check_out_date  
        );  
  
    -- Выводим результаты через уведомления  
    RAISE NOTICE 'Доступные одноместные номера на завтра:';  
  
    FOR r_record IN SELECT * FROM temp_available_rooms  
    LOOP  
        RAISE NOTICE 'Номер: %, Тип: %, Цена: %',  
            r_record.room_number, r_record.room_type, r_record.base_price;  
    END LOOP;  
  
    -- Показываем результаты в табличном виде  
    COMMIT; -- Завершаем транзакцию, чтобы временная таблица была видима  
    EXECUTE 'SELECT * FROM temp_available_rooms';  
END;
```

Рисунок 4 – Скриншот реализации процедуры 2.

Проверка в PSQL:


```

hotel_db=# -- Добавляем одноместный тип номера (если нет)
hotel_db=# INSERT INTO Room_type (name, beds_count, base_price)
hotel_db=# VALUES ('Одноместный стандарт', 1, 2000.00)
hotel_db=# ON CONFLICT DO NOTHING;
INSERT 0 1
hotel_db=#
hotel_db=# -- Добавляем свободный одноместный номер
hotel_db=# INSERT INTO Room (branch_code, room_type_id, number, occupied_status, cleaning_status)
hotel_db=# SELECT 1, room_type_id, 101, false, true
hotel_db=# FROM Room_type WHERE beds_count = 1 LIMIT 1
hotel_db=# ON CONFLICT DO NOTHING;
INSERT 0 1
hotel_db=#
hotel_db=# -- Добавляем цену на текущий период
hotel_db=# INSERT INTO Price (amount, period_start, period_end, room_type_id)
hotel_db=# SELECT 2000.00, CURRENT_DATE, CURRENT_DATE + INTERVAL '1 week', room_type_id
hotel_db=# FROM Room_type WHERE beds_count = 1 LIMIT 1
hotel_db=# ON CONFLICT DO NOTHING;
INSERT 0 1
_

hotel_db=# CALL get_available_single_rooms(1);
ЗАМЕЧАНИЕ:  таблица "temp_available_rooms" не существует, пропускается
ЗАМЕЧАНИЕ:  Доступные одноместные номера на завтра:
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Бизнес, Цена: 2000
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Бизнес, Цена: 2000
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Бизнес, Цена: 2000
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Бизнес, Цена: 2000
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 21788.27
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 41561.89
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 3632.13
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 26434.25
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 13688.11
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 12509.79
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 18871.71
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 14401.01
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 44300.14
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 31490.709999999995
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 47032.69
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 36435.59
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 21668.46
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 25574.59
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 10125.48
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 24469.52
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 2761.29
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 37621.56
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 13737.04
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 46083.84
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 36470.32
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 18755.22
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 23572.9
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 22425.32
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 32747.35
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 22650.74
ЗАМЕЧАНИЕ:  Номер: 101, Тип: Стандарт, Цена: 34747.39
CALL
_

```

Рисунок 5 – Скриншоты проверки процедуры 2.

1.3.3 Процедура бронирования двухместного номера в гостинице на заданную дату и количество дней проживания

```
CREATE OR REPLACE PROCEDURE book_double_room(
    IN p_hotel_id INTEGER,
    IN p_guest_passport VARCHAR,
    IN p_start_date DATE,
    IN p_days INTEGER,
    IN p_employee_id INTEGER,
    OUT p_booking_id INTEGER,
    OUT p_room_number INTEGER,
    OUT p_total_amount DECIMAL(10,2))
LANGUAGE plpgsql
AS $$
DECLARE
    v_room_id INTEGER;
    v_room_type_id INTEGER;
    v_daily_price DECIMAL(10,2);
    v_contract_id INTEGER;
    v_end_date DATE;
BEGIN
    -- Проверяем, что количество дней положительное
    IF p_days <= 0 THEN
        RAISE EXCEPTION 'Количество дней должно быть положительным';
    END IF;

    -- Вычисляем дату выезда
    v_end_date := p_start_date + (p_days - 1) * INTERVAL '1 day';

    -- Ищем доступный двухместный номер
    SELECT r.room_id, r.room_type_id, r.number, p.amount INTO v_room_id, v_room_type_id, p_room_number, v_daily_price
    FROM Room r
    JOIN Room_type rt ON r.room_type_id = rt.room_type_id
    JOIN Price p ON rt.room_type_id = p.room_type_id
    WHERE r.branch_code = p_hotel_id
    AND rt.beds_count = 2 -- Двухместные номера
    AND r.occupied_status = false -- Свободные номера
    AND r.cleaning_status = true -- Чистые номера
    AND p.start_date BETWEEN p.period_start AND p.period_end
    AND NOT EXISTS (
        SELECT 1 FROM Booking b
        WHERE b.room_id = r.room_id
        AND (p.start_date, v_end_date) OVERLAPS (b.check_in_date, b.check_out_date)
    )
    LIMIT 1;

    -- Если номер не найден
    IF v_room_id IS NULL THEN
        RAISE EXCEPTION 'Нет доступных двухместных номеров на указанные даты';
    END IF;

    -- Находим контракт сотрудника
    SELECT c.contract_id INTO v_contract_id
    FROM Contract c
    WHERE c.employee_id = p_employee_id
    LIMIT 1;

    IF v_contract_id IS NULL THEN
        RAISE EXCEPTION 'Сотрудник с ID % не найден или не имеет контракта', p_employee_id;
    END IF;

    -- Рассчитываем общую сумму
    p_total_amount := v_daily_price * p_days;

    -- Создаем бронирование
    INSERT INTO Booking (
        booking_date,
        check_in_date,
        check_out_date,
        total_amount,
        status,
        room_id,
        contract_id,
        passport_number
    ) VALUES (
        CURRENT_DATE,
        p_start_date,
        v_end_date,
        p_total_amount,
        true,
        v_room_id,
        v_contract_id,
        p_guest_passport
    ) RETURNING booking_id INTO p_booking_id;

    -- Помечаем номер как занятый
    UPDATE Room SET occupied_status = true WHERE room_id = v_room_id;
```

Рисунок 6 – Скриншот реализации процедуры 3.

Проверка в PSQL:

```
hotel_db=# -- Проверка последнего созданного бронирования
hotel_db=# SELECT * FROM Booking ORDER BY booking_id DESC LIMIT 1;
 booking_id | booking_date | check_in_date | check_out_date | total_amount | status | room_id | contract_id | passport_number
-----+-----+-----+-----+-----+-----+-----+-----+-----
          180 | 2024-11-17   | 2024-12-05   | 2024-12-11   | 47487.08    | t      |    915 |          98 | 1819 582864
(1 row)

hotel_db=#
hotel_db=# -- Проверка статуса номера
hotel_db=# SELECT r.room_id, r.number, r.occupied_status
hotel_db=# FROM Room r
hotel_db=# JOIN Booking b ON r.room_id = b.room_id
hotel_db=# ORDER BY b.booking_id DESC LIMIT 1;
 room_id | number | occupied_status
-----+-----+-----
       915 |    305 | f
(1 row)

hotel_db=# -- Проверка последнего созданного бронирования
hotel_db=# SELECT * FROM Booking ORDER BY booking_id DESC LIMIT 1;
 booking_id | booking_date | check_in_date | check_out_date | total_amount | status | room_id | contract_id | passport_number
-----+-----+-----+-----+-----+-----+-----+-----+-----
          180 | 2024-11-17   | 2024-12-05   | 2024-12-11   | 47487.08    | t      |    915 |          98 | 1819 582864
```

Рисунок 6 – Скриншот проверки процедуры 3.

1.4 Создание триггеров

1.4.1 Триггер для автоматического обновления статуса номера при бронировании

Назначение: При создании бронирования автоматически пометать номер как занятый.

```
CREATE OR REPLACE FUNCTION update_room_status_on_booking()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Room SET occupied_status = true WHERE room_id = NEW.room_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_after_booking_insert
AFTER INSERT ON Booking
FOR EACH ROW
EXECUTE FUNCTION update_room_status_on_booking();
```

Рисунок 7 – Скриншот реализации триггера 1.

Тестирование:

```

hotel_db=# -- Проверяем текущий статус номера
hotel_db=# SELECT room_id, number, occupied_status FROM Room WHERE occupied_status = false LIMIT 1;
 room_id | number | occupied_status 
-----+-----+-----
      10 |    348 | f
(1 row)

hotel_db=#
hotel_db=# -- Создаем тестовое бронирование
hotel_db=# INSERT INTO Booking (booking_date, check_in_date, check_out_date, total_amount, status, room_id, contract_id, passport_number)
hotel_db=# VALUES (CURRENT_DATE, CURRENT_DATE + 1, CURRENT_DATE + 3, 5000, true,
hotel_db=#          (SELECT room_id FROM Room WHERE occupied_status = false LIMIT 1),
hotel_db=#          (SELECT contract_id FROM Contract LIMIT 1),
hotel_db=#          (SELECT passport_number FROM Guest LIMIT 1));
INSERT 0 1
hotel_db=#
hotel_db=# -- Проверяем, что статус номера изменился
hotel_db=# SELECT room_id, number, occupied_status FROM Room WHERE room_id = [10];
ОШИБКА: ошибка синтаксиса (примерное положение: "(")
LINE 1: ..._id, number, occupied_status FROM Room WHERE room_id = [10];
                                ^
hotel_db=# -- Проверяем, что статус номера изменился
hotel_db=# SELECT room_id, number, occupied_status FROM Room WHERE room_id = 10;
 room_id | number | occupied_status 
-----+-----+-----
      10 |    348 | t
(1 row)

```

Рисунок 8 – Скриншот проверки триггера 1.

1.4.2 Триггер для проверки дат бронирования

Назначение: Проверять, что дата заезда раньше даты выезда.

```

CREATE OR REPLACE FUNCTION validate_booking_dates()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.check_in_date >= NEW.check_out_date THEN
        RAISE EXCEPTION 'Дата заезда должна быть раньше даты выезда';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_before_booking_insert_update
BEFORE INSERT OR UPDATE ON Booking
FOR EACH ROW
EXECUTE FUNCTION validate_booking_dates();

```

Рисунок 9 – Скриншот реализации триггера 2.

Тестирование:

```

hotel_db=# -- Пытаемся создать некорректное бронирование (должно вызывать ошибку)
hotel_db=# INSERT INTO Booking (booking_date, check_in_date, check_out_date, total_amount, status, room_id, contract_id, passport_number)
hotel_db=# VALUES (CURRENT_DATE, CURRENT_DATE + 3, CURRENT_DATE + 1, 5000, true,
hotel_db=#          (SELECT room_id FROM Room LIMIT 1),
hotel_db=#          (SELECT contract_id FROM Contract LIMIT 1),
hotel_db=#          (SELECT passport_number FROM Guest LIMIT 1));
ОШИБКА: Дата заезда должна быть раньше даты выезда
CONTEXT:  функция PL/pgSQL validate_booking_dates(), строка 4, оператор RAISE

```

Рисунок 10 – Скриншот проверки триггера 2.

1.4.3 Триггер для автоматической очистки номера после выезда

Назначение: После изменения даты выезда на прошедшую, пометить номер как требующий уборки.

```
CREATE OR REPLACE FUNCTION mark_room_for_cleaning()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.check_out_date < CURRENT_DATE AND OLD.check_out_date >= CURRENT_DATE THEN
        UPDATE Room SET cleaning_status = false WHERE room_id = NEW.room_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_after_booking_update
AFTER UPDATE ON Booking
FOR EACH ROW
WHEN (OLD.check_out_date IS DISTINCT FROM NEW.check_out_date)
EXECUTE FUNCTION mark_room_for_cleaning();
```

Рисунок 11 – Скриншот реализации триггера 3.

Тестирование:

```
hotel_db=# -- Создаем тестовое бронирование с выездом в будущем
hotel_db=# INSERT INTO Booking (booking_date, check_in_date, check_out_date, total_amount, status, room_id, contract_id, passport_number)
hotel_db=# VALUES (CURRENT_DATE, CURRENT_DATE - 2, CURRENT_DATE + 1, 5000, true,
hotel_db=#         (SELECT room_id FROM Room LIMIT 1),
hotel_db=#         (SELECT contract_id FROM Contract LIMIT 1),
hotel_db=#         (SELECT passport_number FROM Guest LIMIT 1))
hotel_db=# RETURNING booking_id;
 booking_id
-----
      185
(1 row)

INSERT 0 1
hotel_db=#
hotel_db=# -- Меняем дату выезда на прошедшую
hotel_db=# UPDATE Booking SET check_out_date = CURRENT_DATE - 1 WHERE booking_id = [ID бронирования];
ОШИБКА:  ошибка синтаксиса (примерное положение: "[")
LINE 1: ...ck_out_date = CURRENT_DATE - 1 WHERE booking_id = [ID ȲaR-Ea...
                                     ^

hotel_db=#
hotel_db=# -- Проверяем статус уборки номера
hotel_db=# SELECT room_id, number, cleaning_status FROM Room WHERE room_id = 185;
 room_id | number | cleaning_status
-----
      185 |    580 | t
(1 row)
```

Рисунок 12 – Скриншот проверки триггера 3.

1.4.4 Триггер для проверки минимальной стоимости бронирования

Назначение: Запрещать бронирования с общей суммой менее 1000 рублей.

```
CREATE OR REPLACE FUNCTION booking_checks()
RETURNS TRIGGER AS $$
DECLARE
    v_discount DECIMAL(10,2);
BEGIN
    -- Проверка минимальной суммы
    IF NEW.total_amount < 1000 THEN
        RAISE EXCEPTION 'Минимальная сумма бронирования - 1000 рублей';
    END IF;

    -- Применение акций (если нужно)
    SELECT discount_percent INTO v_discount
    FROM Promotion p
    JOIN Room r ON p.room_type_id = r.room_type_id
    WHERE r.room_id = NEW.room_id
        AND CURRENT_DATE BETWEEN p.start_date AND p.end_date
    LIMIT 1;

    IF v_discount IS NOT NULL THEN
        NEW.total_amount := NEW.total_amount * (1 - v_discount/100);
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Удаляем старые триггеры
DROP TRIGGER IF EXISTS trg_before_booking_insert ON Booking;
DROP TRIGGER IF EXISTS trg_min_booking_amount ON Booking;

-- Создаем один объединенный триггер
CREATE TRIGGER trg_booking_checks
BEFORE INSERT ON Booking
FOR EACH ROW
EXECUTE FUNCTION booking_checks();
```

Рисунок 13 – Скриншот реализации триггера 4.

Тестирование:


```

hotel_db=# -- Попытка бронирования на 800 рублей (должна вызвать ошибку)
hotel_db=# INSERT INTO Booking (booking_date, check_in_date, check_out_date, total_amount, status, room_id, contract_id, passport_number)
hotel_db=# VALUES (CURRENT_DATE, CURRENT_DATE + 1, CURRENT_DATE + 2, 800, true,
hotel_db=# (SELECT room_id FROM Room LIMIT 1),
hotel_db=# (SELECT contract_id FROM Contract LIMIT 1),
hotel_db=# (SELECT passport_number FROM Guest LIMIT 1));
ОШИБКА: Минимальная сумма бронирования - 1000 рублей
CONTEXT:  функция PL/pgSQL booking_checks(), строка 7, оператор RAISE
hotel_db=#
hotel_db=# -- Успешное бронирование
hotel_db=# INSERT INTO Booking (booking_date, check_in_date, check_out_date, total_amount, status, room_id, contract_id, passport_number)
hotel_db=# VALUES (CURRENT_DATE, CURRENT_DATE + 1, CURRENT_DATE + 2, 1200, true,
hotel_db=# (SELECT room_id FROM Room LIMIT 1),
hotel_db=# (SELECT contract_id FROM Contract LIMIT 1),
hotel_db=# (SELECT passport_number FROM Guest LIMIT 1));
INSERT 0 1 _

```

Рисунок 14 – Скриншот проверки триггера 3.

1.4.5 Триггер для автоматического создания уборки после освобождения номера

Назначение: Создавать задание на уборку при освобождении номера.

```

CREATE OR REPLACE FUNCTION create_cleaning_task()
RETURNS TRIGGER AS $$
DECLARE
    v_employee_id INTEGER;
    v_contract_id INTEGER;
BEGIN
    IF NEW.occupied_status = false AND OLD.occupied_status = true THEN
        -- Находим любого сотрудника для уборки
        SELECT employee_id INTO v_employee_id
        FROM Employee
        LIMIT 1;

        -- Находим контракт этого сотрудника
        SELECT contract_id INTO v_contract_id
        FROM Contract
        WHERE employee_id = v_employee_id
        LIMIT 1;

        -- Создаем запись об уборке (с правильным порядком столбцов)
        INSERT INTO Cleaning (
            cleaning_date,
            cleaning_status,
            room_id,
            employee_id,
            contract_id
        ) VALUES (
            CURRENT_DATE,
            false,
            NEW.room_id,
            v_employee_id,
            v_contract_id
        );
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trg_after_room_update
AFTER UPDATE ON Room
FOR EACH ROW
WHEN (OLD.occupied_status IS DISTINCT FROM NEW.occupied_status)
EXECUTE FUNCTION create_cleaning_task();

```

Рисунок 15 – Скриншот реализации триггера 5.

Тестирование:

```

hotel_db=# -- 1. Проверяем текущее состояние
hotel_db=# SELECT room_id, number, occupied_status, cleaning_status
hotel_db=# FROM Room WHERE number = 102;
 room_id | number | occupied_status | cleaning_status 
-----+-----+-----+-----
      157 |     102 | f               | t
      619 |     102 | t               | f
      729 |     102 | f               | f
      983 |     102 | t               | f
(4 rows)

hotel_db=#
hotel_db=# -- 2. Освобождаем номер (должен сработать триггер)
hotel_db=# UPDATE Room SET occupied_status = false
hotel_db=# WHERE number = 102;
UPDATE 4
hotel_db=#
hotel_db=# -- 3. Проверяем, что задача на уборку создана
hotel_db=# SELECT c.cleaning_id, c.cleaning_date, c.cleaning_status,
hotel_db=#         r.number as room_number, e.full_name as employee_name
hotel_db=# FROM Cleaning c
hotel_db=# JOIN Room r ON c.room_id = r.room_id
hotel_db=# JOIN Employee e ON c.employee_id = e.employee_id
hotel_db=# WHERE r.number = 102;
 cleaning_id | cleaning_date | cleaning_status | room_number | employee_name 
-----+-----+-----+-----+-----
          143 | 2025-01-14    | f               |          102 | Трэшмэйт Фёухц Фьёэютшү
          436 | 2024-06-06    | f               |          102 | Фьёэют Лэфёхц Лечэхүйтшү
          685 | 2025-03-08    | f               |          102 | Трэшмэйт Лэфёхц Лечэхүйтшү
         1005 | 2025-05-03    | f               |          102 | Лтрэют Фёухц Фьёэютшү
         1006 | 2025-05-03    | f               |          102 | Лтрэют Фёухц Фьёэютшү
(5 rows)

hotel_db=#
hotel_db=# -- 4. Проверяем статус комнаты (должен быть cleaning_status = false)
hotel_db=# SELECT room_id, number, cleaning_status
hotel_db=# FROM Room WHERE number = 102;
 room_id | number | cleaning_status 
-----+-----+-----
      157 |     102 | t
      619 |     102 | f
      729 |     102 | f
      983 |     102 | f
(4 rows)

```

Рисунок 16 – Скриншот проверки триггера 5.

1.4.6 Триггер для проверки возраста гостя

Назначение: Не допускать регистрацию гостей младше 18 лет.


```

CREATE OR REPLACE FUNCTION check_guest_age()
RETURNS TRIGGER AS $$
BEGIN
    IF EXTRACT(YEAR FROM AGE(NEW.birth_date)) < 18 THEN
        RAISE EXCEPTION 'Гость должен быть старше 18 лет';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_before_guest_insert
BEFORE INSERT ON Guest
FOR EACH ROW
EXECUTE FUNCTION check_guest_age();

```

Рисунок 17 – Скриншот реализации триггера 6.

Тестирование:

```

hotel_db=# -- Пытаемся добавить несовершеннолетнего гостя (должно вызвать ошибку)
hotel_db=# INSERT INTO Guest (passport_number, full_name, registration_address, birth_date)
hotel_db=# VALUES ('TE123456', 'Молодой Гость', 'ул. Тестовая, 1', CURRENT_DATE - INTERVAL '17 years');
ОШИБКА: Гость должен быть старше 18 лет
CONTEXT: функция PL/pgSQL check_guest_age(), строка 4, оператор RAISE
hotel_db=#

```

Рисунок 18 – Скриншот проверки триггера 6.

1.4.7 Триггер для автоматического применения акций к бронированию

Назначение: Автоматически применять доступные акции при создании бронирования.

```

CREATE OR REPLACE FUNCTION apply_promotions()
RETURNS TRIGGER AS $$
DECLARE
    v_discount DECIMAL(10,2);
    v_message TEXT;
BEGIN
    -- Ищем активную акцию для данного типа номера
    SELECT discount_percent INTO v_discount
    FROM Promotion p
    JOIN Room r ON p.room_type_id = r.room_type_id
    WHERE r.room_id = NEW.room_id
    AND CURRENT_DATE BETWEEN p.start_date AND p.end_date
    LIMIT 1;

    -- Если акция найдена, применяем скидку
    IF v_discount IS NOT NULL THEN
        NEW.total_amount := NEW.total_amount * (1 - v_discount/100);
        v_message := 'Применена скидка ' || v_discount || '%, итоговая сумма: ' || NEW.total_amount;
        RAISE NOTICE '%', v_message;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

Рисунок 19 – Скриншот реализации триггера 7.

Тестирование:

```
hotel_db=# -- 1. Создаем тестовую акцию
hotel_db=# INSERT INTO Promotion (name, discount_percent, start_date, end_date, room_type_id)
hotel_db=# VALUES ('Тестовая скидка', 10, CURRENT_DATE - 1, CURRENT_DATE + 7,
hotel_db=# (SELECT room_type_id FROM Room_Type LIMIT 1));
INSERT 0 1
hotel_db=#
hotel_db=# -- 2. Создаем тестовое бронирование
hotel_db=# INSERT INTO Booking (booking_date, check_in_date, check_out_date, total_amount, status, room_id, contract_id, passport_number)
hotel_db=# SELECT
hotel_db=#     CURRENT_DATE,
hotel_db=#     CURRENT_DATE + 1,
hotel_db=#     CURRENT_DATE + 3,
hotel_db=#     5000.00,
hotel_db=#     true,
hotel_db=#     r.room_id,
hotel_db=#     (SELECT contract_id FROM Contract LIMIT 1),
hotel_db=#     (SELECT passport_number FROM Guest LIMIT 1)
hotel_db=# FROM Room r
hotel_db=# WHERE r.room_type_id = (SELECT room_type_id FROM Promotion WHERE name = 'Тестовая скидка' LIMIT 1)
hotel_db=# LIMIT 1;
ОШИБКА: Количество гостей превышает вместимость номера
CONTEXT:  функция PL/pgSQL check_room_capacity(), строка 11, оператор RAISE
hotel_db=#
hotel_db=# -- 3. Проверяем, что скидка применилась
hotel_db=# SELECT b.*,
hotel_db=#     b.total_amount AS final_amount,
hotel_db=#     (SELECT discount_percent FROM Promotion
hotel_db=#     WHERE room_type_id = (SELECT room_type_id FROM Room WHERE room_id = b.room_id)
hotel_db=#     AND CURRENT_DATE BETWEEN start_date AND end_date) AS discount
hotel_db=# FROM Booking b
hotel_db=# ORDER BY booking_id DESC
hotel_db=# LIMIT 1;
 booking_id | booking_date | check_in_date | check_out_date | total_amount | status | room_id | contract_id | passport_number | final_amount | discount
-----
(1 row)
```

booking_id	booking_date	check_in_date	check_out_date	total_amount	status	room_id	contract_id	passport_number	final_amount	discount
185	2025-05-03	2025-05-01	2025-05-04	5000	t	2	1	4283 940771	5000	

Рисунок 20 – Скриншот проверки триггера 7.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были успешно освоены практические навыки работы с хранимыми процедурами, функциями и триггерами в СУБД PostgreSQL. Разработанные три процедуры продемонстрировали возможность инкапсуляции сложной бизнес-логики на уровне базы данных, что позволяет повысить производительность приложений за счет уменьшения количества обращений к БД и переноса вычислений на сервер.

Созданные семь оригинальных триггеров реализовали важные аспекты контроля целостности данных и автоматизации бизнес-процессов гостиницы. Триггеры обеспечивают проверку возрастных ограничений для гостей, контроль дат бронирования и уборки, автоматическое применение акций, обновление статусов номеров и создание задач для персонала. Особое внимание было уделено обработке ошибок и созданию информативных сообщений для пользователей.

Практическая работа подтвердила преимущества использования триггеров для поддержания согласованности данных и реализации сложных бизнес-правил на уровне базы данных. Особенно ценным оказалось применение триггеров типа BEFORE для валидации данных и AFTER для реализации каскадных изменений. Также была продемонстрирована эффективность комбинирования процедур и триггеров для создания комплексных решений.

Выполнение дополнительных заданий позволило углубить понимание особенностей работы с переменными в PL/pgSQL, обработкой исключений и транзакциями. Разработанные объекты базы данных полностью соответствуют требованиям варианта задания и готовы к использованию в реальной системе управления гостиницей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация PostgreSQL [Электронный ресурс] // Официальный сайт PostgreSQL. 1996-2025. URL: <https://www.postgresql.org/docs/13/index.html> (дата обращения: 20.03.2025).
2. Документация pgAdmin 4 PostgreSQL [Электронный ресурс] // Официальный сайт pgAdmin. URL: <https://www.pgadmin.org/docs/pgadmin4/latest/> (дата обращения: 21.03.2025)